ALGORITHMES DISTRIBUÉS POUR SYSTÈMES DISTRIBUÉS

par

Stéphane Rivard

mémoire présenté au Département de mathématiques et d'informatique
en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, septembre 1998

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-40619-9

Canada

Le 11 septembre 1998, le jury suivant a accepté ce mémoire dans sa version finale.
date

Président-rapporteur: M. Abdelhamid Benchakroun _____
Département de mathématiques et d'informatique

Membre: M. Djemel Ziou _____
Département de mathématiques et d'informatique

Membre: M.Béchir Ayeb _____
Département de mathématiques et d'informatique

SOMMAIRE

Les systèmes industriels complexes amènent un défi très intéressant au niveau du diagnostic distribué. Pour situer ce travail par rapport à de tels systèmes, nous retenons les mots-clés suivants : algorithmes distribués, alarmes, capacités de communication, fautes, systèmes à propagation d'erreur et consistance du diagnostic.

Pour arriver à concevoir des algorithmes de diagnostic distribué, certains éléments devront préalablement être introduits. Les notions fondamentales sont la modélisation des systèmes industriels, le protocole de communication et différentes structures de données nécessaires pour assurer la consistance et la validité du diagnostic. Bien que plusieurs modèles différents existent pour représenter les systèmes, celui qui sera présenté dans ce mémoire prendra la forme d'un graphe orienté. L'orientation du graphe sera utilisée seulement pour indiquer le sens de la contamination du système; les noeuds pourront communiquer de manière "duplex".

Après avoir établi les notions générales, il sera possible d'introduire des algorithmes de diagnostic. Par raffinement progressif, nous introduirons graduellement trois algorithmes qui nous permettront de nous adapter progressivement à la réalité des systèmes industriels complexes. Pour amorcer le processus, nous adopterons une approche intuitive. Par

la suite, nous poursuivrons la démarche en présentant la relation "happened before". Cette relation permettra d'améliorer la solution précédente, mais elle est difficilement adaptable pour des systèmes de grande taille. Pour remédier à ce problème, nous introduirons la relation de causalité. Cette relation étant plus puissante, il sera possible de développer des algorithmes dynamiques qui respectent les contraintes réelles imposées par les systèmes industriels.

REMERCIEMENTS

Un travail de recherche tel que la réalisation de ce mémoire ne s' effectue pas seul. C'est pour cette raison que je profite de l'occasion qui m'est offerte de remercier les gens qui ont, de près ou de loin, participé à la réalisation de ce mémoire.

Je dis merci à M. Béchir el Ayeb, mon directeur de recherche, pour sa disponibilité, sa patience et son intérêt contagieux pour la recherche.

Je remercie aussi les membres du jury qui ont évalué ce mémoire. Merci à vous pour avoir consacré temps et énergie à l'élaboration de la version finale de ce document.

Je m'en voudrais d'oublier mes parents et amis. Merci à mon père André, à ma mère Fernande, à ma soeur Sylvie et à mon frère Sylvain, qui ont su m'écouter et m'encourager dans les moments les plus difficiles. À mes amis, je tiens aussi à exprimer ma gratitude. Merci à André, Tan Bao et Gilles pour leurs encouragements et les discussions de tous ordres.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

INTRODUCTION

Les systèmes où l'on tente d'identifier les causes d'une faute à partir de symptômes (une alarme, par exemple) sont très étudiés : que l'on pense à l'industrie chimique [6, 8, 19] ou au diagnostic médical [17]. Dans le cadre de cette recherche, nous visons la conception d'une approche distribuée pour la surveillance et le contrôle des systèmes industriels complexes. Le problème que nous posons peut se décrire ainsi. La plupart des systèmes de contrôle et de surveillance sont centralisés. Il en résulte alors d'énormes contraintes et un goulot d'étranglement est quasi inévitable. Le temps de réponse étant contraint, la mise en oeuvre des procédures élaborées pour la sécurité et l'identification de fautes devient très difficile, voire impossible.

En proposant une approche distribuée, nous désirons éliminer la congestion causée par le flux de contrôle en permettant à plusieurs composants de s'autosurveiller. Plus précisément, chaque composant effectue la collecte et le filtrage des données avant de calculer et de propager sa vue locale de l'état du système. La construction de l'état global du système s'effectue par l'échange de messages couvrant les diverses vues locales. Bien entendu, il faut porter une attention particulière au nombre de messages échangés ainsi qu'à la validité et à la consistance de l'état global du système, trois contraintes majeures dans l'élaboration d'une approche distribuée.

Afin de réaliser ce diagnostic, nous modéliserons le système par un graphe et nous tenterons d'identifier les causes d'erreur par des algorithmes distribués. Nous définirons le graphe représentant le système par des composants (les noeuds) et par des liens de propagation d'erreurs (les arcs). De plus, les composants peuvent être munis d'alarmes (voyant lumineux, sonnerie, etc.) afin de manifester la présence de conditions d'exploitation inacceptables ou de composants défectueux. En observant la topologie du système, on remarquera que les algorithmes qui seront développés serviront à minimiser le nombre de tests exécutés sur le système afin d'identifier le composant fautif. De plus, on remarquera que la localisation des alarmes est primordiale pour faciliter la tâche des algorithmes de diagnostic. Toutefois, dans le cadre de cette recherche, nous nous intéresserons seulement aux problèmes reliés à la phase de pré-traitement. Nous ferons abstraction de la partie s'occupant du placement des alarmes ainsi que du traitement permettant de discriminer entre les composants probablement fautifs.

Pour alléger le diagnostic, nous poserons certaines hypothèses concernant le système à diagnostiquer. Premièrement, nous considérerons que le graphe ne contient aucune boucle. La seconde hypothèse concerne le délai de propagation entre deux composants. Pour simplifier le traitement, nous supposerons que le temps pour qu'une faute se propage d'un noeud vers un successeur est nul. À notre système, nous devrons finalement ajouter des facilités de communication. Ainsi, pour chaque lien de propagation d'erreur, nous supposerons que l'on dispose d'un lien de communication "duplex". Ces liens de communication serviront à l'échange de l'information nécessaire au bon fonctionnement des algorithmes de diagnostic.

Puisque chaque noeud possède une capacité de communication avec ses voisins immédiats,

nous ajouterons une capacité de traitement à chaque composant afin de distribuer le diagnostic. Cette capacité de traitement est nécessaire puisque chaque noeud n'a qu'une vue locale du système. En utilisant un certain protocole de communication, nous tenterons d'obtenir un résultat valide et consistant. On définira la consistance par la relation suivante : $\forall i \; \forall j, \; TAB_i = TAB_j$ où i et j sont deux composants du système et TAB_i, TAB_j représentent respectivement l'état des composants du système au noeud i et au noeud j. Le résultat sera consistant si les noeuds obtiennent le même diagnostic du système. Pour exprimer la validité, on dira que le résultat est valide si et seulement si chaque composant non fautif est diagnostiqué comme tel et chaque noeud fautif est déclaré fautif. Ainsi, la validité du diagnostic est définie par les deux relations suivantes :

1. si un noeud est fautif, alors son état est dit non OK (noté NOK) ;
2. si un noeud n'est pas fautif, alors son état est dit OK (noté OK).

En regard des fautes, certaines hypothèses seront aussi posées. Tout d'abord, nous traiterons seulement les fautes à cause unique. De plus, un élément fautif n'a pas la capacité d'interrompre le processus de passage des messages ou d'altérer ceux-ci. Par cette hypothèse, nous nous assurons de la capacité continue du réseau à pouvoir communiquer puisqu'aucune restriction n'est posée en regard du degré de connexité du système.

La suite de ce mémoire sera présentée de la manière suivante. Le premier chapitre introduira certaines définitions ainsi que les structures de données nécessaires au bon fonctionnement des algorithmes. Le deuxième chapitre donnera une solution intuitive du problème de diagnostic. Le troisième chapitre exprimera un algorithme en utilisant la relation *happened before* alors que le quatrième chapitre présentera la possibilité d'utiliser l'histoire causale pour identifier les composants fautifs. Le cinquième chapitre présentera un cas

sur lequel les trois algorithmes seront appliqués. Finalement, la conclusion permettra de faire une synthèse de ce mémoire et d'orienter les travaux futurs.

CHAPITRE 1

PRÉLIMINAIRES

La première étape que l'on rencontre lorsque l'on souhaite étudier un système, est la construction d'un modèle représentant ledit système. *A priori*, plusieurs modèles peuvent être utilisés. Pour s'en convaincre, il suffit de regarder dans la littérature pour identifier des méthodes telles que : les systèmes à base de connaissance [6], l'approche topologique [9], l'approche abductive [3, 4], l' approche logique [2] et les arbres [11]. Bien que les approches précédentes permettent de solutionner certains problèmes, le modèle le plus couramment rencontré utilise la notation des graphes [1, 7, 13, 14, 15]. Ainsi, ce chapitre introduira un modèle topologique basé sur une représentation par graphe, les structures de données présentes à chaque composant du système ainsi qu'un protocole de communication nécessaire au bon fonctionnement des algorithmes.

1.1 Définition du modèle

Soit $\mathcal{S}$, un système avec propagation d'erreur pouvant être détecté par des senseurs et se manifester par des alarmes. Nous définirons le modèle de $\mathcal{S}$ par un graphe orienté $\mathcal{G}$

qui se compose du couple (V, E). Ainsi, $\mathcal{G}$ = (V, E) où V est l'ensemble des composants du système (noeuds) et E représente l'ensemble des arcs. Nous exprimerons un arc par rapport aux noeuds situés à ses extrémités en tenant compte de l'orientation de l'arc. Soit l'arc (u, v) $\in E$ avec u et $v \in V$. Nous dirons qu'une faute se propagera de u vers v par le lien qui les unit. En ce qui concerne le nombre de noeuds et d'arcs, nous poserons que $\mid V \mid$ = n et $\mid E \mid$ = e. Malgré le fait que le délai de propagation entre les noeuds du système soit nul, nous introduirons tout de même t_{uv} comme étant le temps de propagation entre deux noeuds. Puisque le système est représenté par un graphe, chaque noeud peut avoir des prédécesseurs et des successeurs. Ainsi, nous identifierons respectivement $\Gamma^{-1}(x)$ et $\Gamma(x)$ comme étant l'ensemble des prédécesseurs immédiats de x et l'ensemble des successeurs immédiats de x. Nous généraliserons en posant $\Gamma^*(x)$ l'ensemble de tous les successeurs de x et $(\Gamma^{-1})^*(x)$ l'ensemble de tous les prédécesseurs de x .

Bien que les notions de prédécesseur et successeur soient déjà introduites, elles ne sont pas encore définies. Puisque l'on est dans un graphe orienté et qu'aucune boucle ne peut être présente dans le système, il sera possible de définir les prédécesseurs et les successeurs de manière absolue. Les équations suivantes permettront d'identifier respectivement l'ensemble des prédécesseurs immédiats (équation 1.1) et l'ensemble de tous les antécédents d'un noeud (équation 1.2).

$$\Gamma^{-1}(x) = \{y \in V \ / \ (y, x) \in E\} \tag{1.1}$$

$$(\Gamma^{-1})^*(x) = \left\{y \in \ (\Gamma^{-1}(x) \cup (\Gamma^{-1})^*(y)) \ \forall y \in \Gamma^{-1}(x)\right\} \tag{1.2}$$

Afin de définir la notion de successeur nous utiliserons l'équation 1.3 pour les successeurs

immédiats et l'équation 1.4 pour l'ensemble de tous les successeurs.

$$\Gamma(x) = \{y \in \ V \ / \ (x, y) \in E\} \tag{1.3}$$

$$\Gamma^*(x) = \{y \in \ (\Gamma(x) \cup \Gamma^*(y)) \ \forall y \in \Gamma(x)\} \tag{1.4}$$

On peut remarquer que le système est composé de deux types de composants. Ceux équipés d'alarme qui seront notés SEO (*Sensor Equiped Object*) et ceux ne permettant pas de détecter des conditions inacceptables de fonctionnement, les NSO (*Non Sensor Object*). À partir du groupe des composants équipés d'alarme, il est possible d'obtenir deux sous ensembles de V. Le premier, noté A, représente l'ensemble des noeuds équipés d'alarme. Nous poserons $\mid A \mid$ = k. Le second sous-ensemble, A_R, représente l'ensemble des alarmes bruyantes du système et $\mid A_R \mid = k_R$. De plus, on peut remarquer la relation suivante : $A_R \subseteq A \subseteq V$. Tant et aussi longtemps que $A_R = \phi$, c'est qu'aucune faute n'est présente dans le système. Dès que cette condition n'est plus respectée, c'est que certains composants du système opèrent sous des conditions inacceptables. En admettant que certains noeuds peuvent éventuellement se manifester, nous introduisons le concept de faute. Mais avant d'identifier le composant fautif (nommé F), nous passerons par un ensemble intermédiaire : l'ensemble des fautifs potentiels qui sera nommé P_S.

Afin de représenter le système que nous souhaitons modéliser, nous utiliserons la notation de Coad et Yourdon [12]. Dans cette notation, les objets sont représentés par des boîtes de forme rectangulaire divisées en trois parties. La première identifie le nom de l'objet, la seconde les attributs et la troisième les méthodes associées à l'objet ou les messages auxquels l'objet répond. En plus d'identifier les objets, leurs attributs et leurs

méthodes, le modèle de Coad Yourdon permet d'identifier les relations d'assemblage et de classification entre les objets en relation.

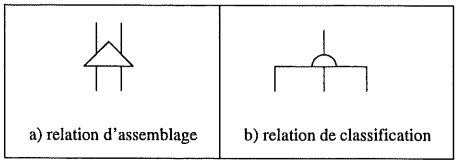

Figure 1.1: *Types de relations*

La figure 1.1 illustre les deux types de relations d'un modèle alors que la figure 1.2 représente un modèle objet sommaire du système étudié. En consultant la figure 1.3, il est possible d'identifier certains ensembles précédemment définis. Ainsi, l'ensemble des noeuds du système est donné par V = { a, b, c, d, e, f, g, h, i, j, k, l, m }, l'ensemble des alarmes par A = { f, g, h, i, j, m } et l'ensemble des alarmes bruyantes par A_R = { g, h, m }. Nous ferons abstraction des ensembles E (arcs), $\Gamma(noeud)$, $\Gamma^{-1}(noeud)$ $\Gamma^*(noeud)$ et $(\Gamma^{-1})^*(noeud)$ en citant un seul exemple pour chacun d'eux. Par exemples, $\Gamma(g)$ = { k }, $\Gamma^{-1}(g)$ = { b, c}, $\Gamma^*(g)$ = { k, m }, $(\Gamma^{-1})^*(g)$ = { a, b, c } alors que l'arc reliant les noeuds a et b sera noté (a, b).

1.2 Capacités de communication

En distribuant le diagnostic à l'ensemble des composants du système, il faut reconnaître l'importance de pouvoir propager tout changement d'état dudit système afin de conserver la validité et la consistance du diagnostic. Pour ce faire, nous devrons déterminer de quelle manière chaque noeud communiquera avec ses voisins. Par manière de communiquer, nous faisons référence aux messages compris par l'ensemble des noeuds du système. Ce langage commun, le protocole, sera développé en faisant certaines hypothèses.

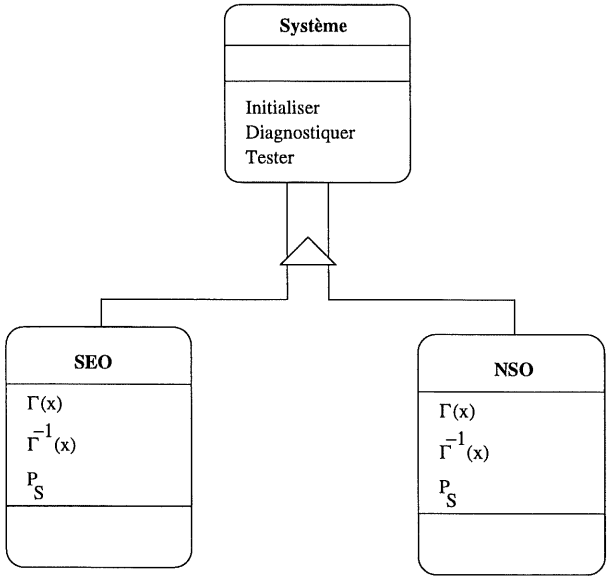

Figure 1.2: *Modèle objet du système*

Bien qu'aucun noeud ne puisse refuser de communiquer, rien ne garantit la validité du message. Pour contrer cette possibilité, il est nécessaire de poser que les noeuds ainsi que les liens n'ont pas la capacité d'altérer le message qu'ils transmettent et propagent. Puisque chaque message envoyé se rend à destination et que son contenu n'a pas été modifié ou détruit, il n'est pas nécessaire de confirmer chaque message par un accusé de réception.

Pour définir le protocole, on donnera simplement une description des messages pouvant être compris par l'ensemble des noeuds. Tout d'abord, voici les messages qui composent le protocole:

- Notifier alarme (NA);
- Mettre en accusation (MA);

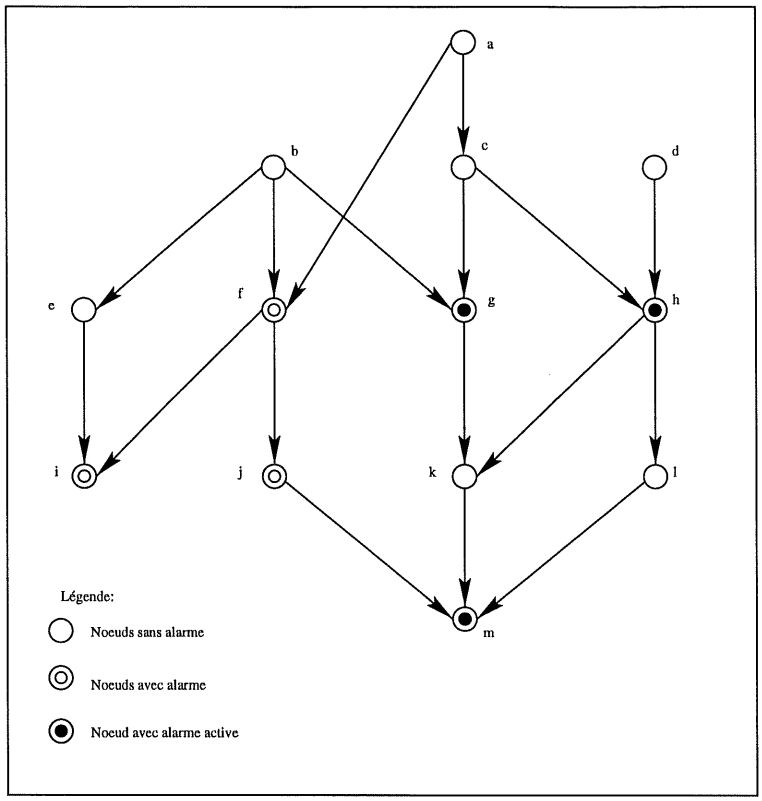

Figure 1.3: *Représentation d'un système*

- Rechercher un alibi (RA);
- Innocenter les descendants (ID);
- Innocenter les ancêtres (IA);
- Propager l'état d'un noeud (PE);
- Rechercher couverture (RC).

Le tableau 1.1 présente les différents messages composant le protocole ainsi que leur signification.

Message	Signification
$< NA, m >$	Le composant m a détecté une faute (alarme active).
$< AR, m >$	Le composant m amorce le processus permettant d'identifier les composants expliquant $A_{R/}$.
$< MA, m >$	Le noeud m accuse ses ancêtres d'être fautifs.
$< RA, m >$	Le noeud m tente de trouver un descendant avec alarme inactive.
$< ID, m >$	Le noeud m avise ses descendants qu'ils ne sont pas fautifs.
$< IA, m >$	Le noeud m avise ses ancêtres qu'ils ne sont pas fautifs.
$< PE, n, m, e >$	Le noeud n est dans l'état e selon le composant m.
$< RC, m >$	Le noeud m recherche une alarme active.

Tableau 1.1: *Protocole de communication*

Il faut remarquer que le premier message sera compris uniquement par les objets de type SEO. Il est évident qu'une alarme ne peut pas être détectée par les composants dépourvus de senseurs. Ainsi, on peut constater que les objets de type NSO pourront réagir aux messages MA, RA, ID, IA, PE, et RC alors que les composants de type SEO pourront répondre aux mêmes messages en plus du message NA.

Après avoir défini la capacité d'un composant à envoyer des messages, il faut s'assurer que les récipiendaires puissent les conserver jusqu'à leur traitement. Pour réaliser cette tâche, une file d'attente de type FIFO (premier arrivé, premier servi) sera utilisée.

1.3 Structures de données

Lorsque l'on veut distribuer le processus de diagnostic, c'est qu'il n'y a pas d'observateur central connaissant l'état général du système. Afin de réaliser un diagnostic consistant et cohérent, il est nécessaire d'introduire certaines structures de données permettant d'emmagasiner localement le nouvel état du système ainsi que certaines connaissances

dérivées facilitant le bon fonctionnement des algorithmes de diagnostic. Pour classer ces structures de données, trois classes sont utilisées :

1. une structure de données représentant l'état général du système ;
2. un certain nombre d'ensembles dérivés de la connaissance sur l'état général du système ;
3. les structures de communication.

La première structure de données qui est définie est celle décrivant l'état du système. Plutôt que d'avoir un observateur central contrôlant toute l'information sur l'état du système, nous placerons une copie de cette structure à chacun des noeuds du système. Ainsi, TAB_i fera référence à la structure de données présente au noeud i.

Identification	*Ring*	*Ringing*	état
a	Non	Non	OK
b	Non	Non	OK
c	Non	Non	OK
d	Non	Non	OK
e	Non	Non	OK
f	Oui	Non	OK
g	Oui	Non	OK
h	Oui	Non	OK
i	Oui	Non	OK
j	Oui	Non	OK
k	Non	Non	OK
l	Non	Non	OK
m	Oui	Non	OK

Tableau 1.2: *Structure de données TAB*

Le tableau 1.2 montre la structure de données qui se retrouvera à chacun des noeuds pour le système modélisé par la figure 1.3 avant qu'il ne se produise une faute. La première colonne correspond à l'identification des noeuds du système. La seconde identifie

les composants équipés d'une alarme par un oui et ceux dépourvus d'un tel dispositif par non. Ces deux premières colonnes ne changeront pas d'état au fur et à mesure du processus de diagnostic. Par contre, les colonnes trois et quatre verront leur contenu être modifiés lors de l'application des algorithmes de diagnostic. La troisième colonne reflète la capacité de certains noeuds d'exprimer des conditions d'exploitation inacceptable. Par rapport à ce qui fut présenté précédemment, cette troisième colonne permet d'identifier l'ensemble des alarmes bruyantes du système (précédemment nommé A_R). Les éléments de l'ensemble A_R seront les composants pour lesquels un oui apparaîtra en troisième colonne de la structure TAB. La dernière colonne du tableau correspond à l'état de chaque composant. L'état d'un composant peut prendre seulement deux valeurs: OK ou NOK. L'ensemble des éléments dont l'état est NOK, correspond à l'ensemble P_S.

En ce qui concerne les structures de communication, nous identifions l'ensemble des prédécesseurs et l'ensemble des successeurs. L'ensemble des prédécesseurs et l'ensemble des successeurs sont utilisés afin de connaître les noeuds avec lesquels un noeud peut communiquer. Ces deux ensembles sont donc présents à chaque noeud et leurs éléments dépendent des liens existants entre les composants du système. La figure 1.4 illustre la décomposition d'un système, les messages auxquels un objet peut répondre ainsi que les attributs des objets.

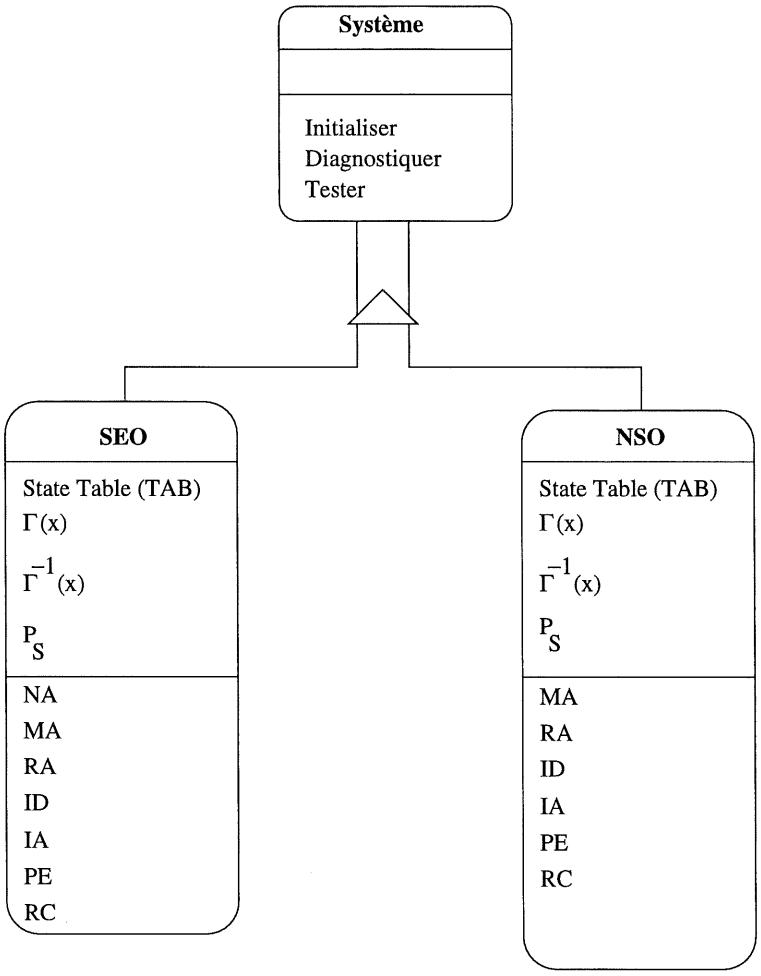

Figure 1.4: *Modèle complet du système*

CHAPITRE 2

SOLUTION INTUITIVE

Le système sur lequel le processus de diagnostic sera appliqué est modélisé par un graphe. L'approche proposée dans cette section se compose de deux grandes étapes: identifier P_S et par la suite restreindre le nombre d'éléments de cet ensemble aux composants ne pouvant pas expliquer A_R. Pour expliquer et évaluer l'approche proposée, le chapitre sera divisé en trois parties: le contexte, l'algorithme et un exemple. La section situant le contexte (section 2.1) permettra d'établir les critères pour lesquels un composant fera ou ne fera pas partie de l'ensemble P_S. La seconde section présentera un algorithme de diagnostic composé de différents modules et respectant les notions contenues dans la partie définissant le contexte (section 2.1). Par la suite, la dernière section donnera les résultats intermédiaires obtenus par l'application de l'algorithme 2.1.

2.1 Contexte

Supposons qu'au temps t, le système soit stable, i.e. qu'il n'y ait aucune alarme bruyante. À ce moment, $A_R = \phi$ et la structure de données (nommée TAB et présenté à la section

1.3) est complète et cohérente. Ainsi, l'état de chaque noeud est OK. Après un certain temps (Δt), à l'instant $t + \Delta t$, une faute se produit. Suite à la contamination du système (propagation d'erreur instantanée), certains noeuds munis d'alarme se manifestent. La tâche des deux algorithmes présentés dans cette section est d'identifier l'ensemble des noeuds permettant d'expliquer la totalité des alarmes bruyantes.

Un noeud x est potentiellement fautif ($x \in P_S$) s'il est incapable de se trouver un alibi et il n'est pas fautif ($x \notin P_S$) s'il peut se trouver un alibi. Pour avoir un alibi, un noeud doit se trouver dans l'un des quatre cas suivants:

1. Cas 1: x est muni d'une alarme qui ne se manifeste pas;
2. Cas 2: $\exists\ y \in (\Gamma^{-1})^*\ (x)$ tel que y est muni d'une alarme active;
3. Cas 3: $\exists\ y \in \Gamma^*(x)$ tel que y est muni d'une alarme inactive;
4. Cas 4: x est incapable de couvrir toutes les alarmes bruyantes (A_R).

Le premier cas est trivial. En ce qui concerne le second, on remarque que la présence d'un noeud prédécesseur ayant une alarme active suffit pour donner un alibi à un noeud. Intuitivement, un noeud z (z étant un successeur de x) ne peut pas être tenu responsable pour l'alarme bruyante du composant x. Dans l'éventualité où un noeud est incapable de se trouver un alibi par les deux cas précédents, il existe un troisième moyen pour le faire. Ainsi, dès qu'un noeud trouve un de ses successeurs avec une alarme inactive, il peut dire qu'il possède un alibi. Il est cohérent de poser ce dernier cas puisqu'un noeud fautif provoquera la manifestation de toutes les alarmes successives à ce noeud. En ce qui concerne le dernier cas, il faut remarquer qu'un composant fautif provoquera la manifestation de toutes les alarmes subséquentes à ce noeud.

2.2 Première solution

En conjuguant le protocole de communication et les possibilités pour un noeud de se trouver un alibi, nous obtenons un premier algorithme. Cet algorithme, nommé solution intuitive, est constitué de cinq étapes (voir algorithme 2.1). La première étape (détaillée à la section 2.2.1) consiste à propager l'identité des composants bruyants, et à identifier les noeuds qui sont incapables de se trouver un alibi par les trois premiers cas présentés à la section précédente. Par la suite, les deux étapes suivantes peuvent être exécutées localement à chacun des noeuds. Le calcul de $A_{R\prime}$ est exécuté afin d'éliminer les alarmes bruyantes déjà couvertes par un noeud prédécesseur ayant lui aussi une alarme bruyante. En remplaçant A_R par $A_{R\prime}$, il est possible qu'il soit nécessaire de supprimer certains éléments de P_S. Bien que le calcul de $A_{R\prime}$ et la mise à jour de P_S ne soient pas nécessaires, ils sont exécutés de manière à minimiser le nombre de recherches requises afin de couvrir l'ensemble restreint des alarmes bruyantes ($A_{R\prime}$), tâche exécutée par le deuxième algorithme (algorithme présenté à la section 2.2.2).

Étendre P_S
Calculer $A_{R\prime}$
Calculer P_S
Restreindre P_S
Si $P_S = \phi$
    Aucune faute simple n'explique $A_{R\prime}$
Sinon
    Pour chaque élément x de l'ensemble P_S faire un test pour s'assurer qu'il est fautif et identifier F

Algorithme 2.1 — Solution intuitive

Pour calculer $A_{R\prime}$ nous nous servons de TAB_i, pour chaque composant i de l'ensemble V. Puisque la structure de données TAB est consistante pour chaque noeud du système, il est possible de calculer $A_{R\prime}$ de manière locale à chaque composant. Le résultat obtenu sera

le même pour tous les noeuds du système. L'ensemble restreint des alarmes bruyantes ($A_{R\prime}$) est calculé par l'équation 2.1.

$$A_{R\prime} = \{x \in \text{ V tel que x.}\textit{Ringing} = \text{ OUI et x.état } = \text{ NOK}\} \tag{2.1}$$

Les noeuds qui sont inclus dans A_R et ceux qui ne le sont pas dans $A_{R\prime}$ correspondent aux composants respectant le cas 2 de la section 2.1. Puisque qu'un composant fautif provoquera toutes les alarmes subséquentes, il n'est pas nécessaire de couvrir toutes ces alarmes mais seulement celles de premier niveau. Après avoir calculé $A_{R\prime}$, il est possible que certains candidats (composants $\in P_S$) ne puissent plus être retenus potentiellement fautifs. Maintenant que $A_{R\prime}$ est calculé, il est possible d'appliquer l'équation 2.2 à l'ensemble des fautifs potentiels:

$$P_S = \begin{cases} \{x \in \text{V tel que x.état} = \text{NOK}\} - A_{R\prime} & \text{si } \mid A_{R\prime} \mid > 1 \\ \{x \in \text{V tel que x.état} = \text{NOK}\} & \text{sinon} \end{cases} \tag{2.2}$$

Puisque $A_{R\prime}$ représente un ensemble de noeuds ayant une alarme bruyante et que chaque composant ne peut pas couvrir un autre élément de cet ensemble, il est légitime d'appliquer la restriction présenté par l'équation 2.2. À partir du moment où il y a plus d'un élément dans l'ensemble $A_{R\prime}$, la mise à jour effectuée à l'ensemble P_S vise essentiellement à réduire le nombre de messages propagés dans le système.

2.2.1 Algorithme Étendre P_S

La première partie de l'algorithme 2.1 consiste à l'exécution de l'algorithme Étendre P_S. Puisque le processus de diagnostic est déclenché, c'est qu'une faute s'est produite dans le

système (il y a au moins une alarme qui se manifeste). Pour initialiser l'algorithme, il faut supposer qu'il existe un mécanisme permettant de générer un message de la forme < NA, x > vers l'objet SEO répondant à ce senseur. Pour la suite de la section, nous présenterons tout d'abord un tableau permettant d'identifier les différents messages auxquels chacun des objets peut répondre, nous donnerons les messages engendrant les transitions d'états des objets du système avant de terminer avec le comportement des objets en fonction des différents messages qu'ils comprennent.

Le modèle du système étant constitué de deux types de composants (NSO, SEO) pouvant prendre chacun deux états (OK, NOK), il existe des traitements différents pour certains messages en rapport au type de composant. Ainsi, il est important d'identifier quels sont les messages auxquels les noeuds ne doivent pas répondre en plus de présenter les événements qui sont impossibles à réaliser. Le tableau 2.1 illustre les situations selon lesquelles un composant doit réagir, ne doit pas réagir ou une situation impossible.

type	NSO	faulty NSO	SEO	faulty SEO
message				
NA	impossible	impossible	réagit	ne réagit pas
MA	réagit	ne réagit pas	ne réagit pas	réagit
ID	réagit	réagit	ne réagit pas	réagit
IA	réagit	réagit	ne réagit pas	impossible
PE	réagit	réagit	réagit	réagit
RA	réagit	ne réagit pas	réagit	ne réagit pas

Tableau 2.1: *Messages traités par l'algorithme Étendre* P_S

Après avoir identifié les messages auxquels les composants peuvent répondre, il faut regarder quels sont les messages qui permettent à un noeud de changer d'état. Pour illustrer les transitions d'états, deux figures seront utilisées. La figure 2.1 permet d'identifier les

messages affectant les composants de type SEO alors que la figure 2.2 illustre les transitions pour les objets de type NSO.

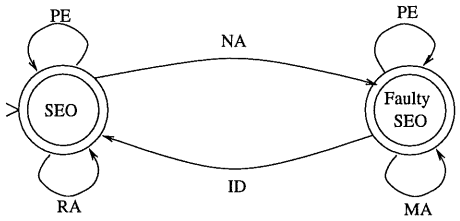

Figure 2.1: *Transition d'états pour un objet SEO*

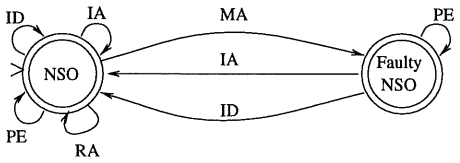

Figure 2.2: *Transition d'états pour un objet NSO*

Maintenant que les transitions d'état sont connues et que l'ensemble des messages auxquels les composants répondent sont identifiés, il reste à justifier les situations impossibles, les situations où aucun traitement n'est requis et les situations exigeant un traitement. Nous présenterons premièrement les justifications pour les cas ne pouvant pas survenir. Par la suite, nous définirons le comportement des composants répondant à l'ensemble des messages avant de terminer en présentant les cas où un composant ne doit pas réagir.

En regardant le tableau 2.1, on remarque que trois cas ne peuvent pas survenir. Pour expliquer les deux premiers cas, il est clair que seuls les composants de type SEO peuvent recevoir le message $< NA, x >$. Le message $< NA, x >$ est donc incompatible avec les composants de type NSO.

Le cas d'un composant SEO recevant un message de type $< IA, x >$ s'explique de la manière suivante. La réception du message $< IA, x >$ par un noeud y identifie un descendant de type SEO. De plus, nous avons précédemment posé que la détection d'une faute par un objet SEO provoquera une réaction de tous les composants de type SEO. La contradiction est claire, il est impossible de trouver un composant de type *faulty SEO* recevant un message de la forme $< IA, x >$.

Lorsqu'une faute se produit, les premiers objets à réagir sont les composants de type SEO. Leur réaction en rapport à la détection d'une faute par au moins un de leurs senseurs se divise en quatre étapes. Tout d'abord, il faut mettre à jour la structure de données TAB locale au composant mis en cause en modifiant les deux champs concernés i.e., *ringing* et *état*. L'objet recevant le message NA passe ainsi de l'état SEO à l'état *Faulty SEO*. Par la suite, il faut accuser les prédécesseurs du noeud courant d'être responsable de la faute au composant x. Il faut noter que le caractère $\#$ sera utilisé pour représenter le noeud courant. La dernière action à entreprendre consiste à propager le changement d'état du noeud x. On remarque que la détection d'une faute par un composant SEO entraîne un changement d'état de ce noeud. Il passe à l'état *Faulty SEO*. Il est clair qu'il est nécessaire de traiter seulement le cas de la manifestation du premier senseur pour un certain composant. Le comportement d'un objet SEO lors d'un événement NA est présenté par l'algorithme 2.2.

En regardant les actions à entreprendre à la suite d'un événement de type NA, on remarque l'envoi d'un message de la forme $< MA, x >$. Certains composants doivent réagir à la réception de ce message alors que pour d'autres, il n'est pas nécessaire d'entreprendre d'actions. En regardant la figure 2.2, on remarque que les composants de type NSO changent d'état à la réception d'un message de la forme $< MA, x >$. Ce changement

Object Type	SEO
Event	NA
Action	1. $TAB_{\#,\#}$.ringing = Vrai; 2. $TAB_{\#,\#}$.état = NOK; 3. Send $< MA, \# >$ To $\Gamma^{-1}(\#)$; 4. Send $< PE, \#, \#, NOK >$ To $\Gamma^{-1}(\#)$, $\Gamma(\#)$.

Algorithme 2.2 — Détection d'une faute

d'état exprime la possibilité pour un composant de type NSO d'être le responsable de la défaillance d'un objet SEO (*Faulty SEO).*

Il est facile de ressortir deux cas où il n'est pas nécessaire pour un composant de réagir à un message $< MA, x >$. Le premier cas provient d'un composant SEO. Puisqu'il est impossible pour un composant SEO d'expliquer une alarme à un composant compris parmi l'ensemble de tous ses descendants un tel composant n'a aucune action à entreprendre. Le deuxième cas est tout aussi trivial. Un composant *Faulty NSO* est dans cet état à la suite de la réception d'un message de la forme $< MA, x >$. Puisque les actions nécessaires ont déjà été entreprises, il serait inutile de les reprendre. Malgré que les composants précédemment identifiés ne doivent pas entreprendre d'actions, certains autres peuvent réagir.

Les objets devant réagir sont regroupés en deux groupes, les objets de type NSO et les les objets de type *Faulty SEO.* Les actions à entreprendre sont différentes pour chacun des deux types d'objet puisque les actions précédemment entreprises par un objet NSO sont différentes par rapport à celles d'un objet *Faulty SEO.* Pour le premier type d'objet (NSO), la réception du message $< MA, x >$ provoquera premièrement un changement d'état puisqu'il est possible qu'il explique le malfonctionnement du composant x. Il en-

treprendra par la suite trois démarches en parallèles: la recherche d'un objet SEO parmi ses descendants (ce qui lui donnerait un alibi), la recherche d'un ancêtre ne possédant pas d'alibi (pour identifier tous les composants pouvant expliquer l'alarme au composant x) et finalement la propagation de son changement d'état (algorithme 2.3).

Object Type	NSO
Event	Receive $< MA, x >$
Action	1. $TAB_{\#,\#}$.état = NOK; 2. Send $< RA, \# >$ To $\Gamma(\#)$; 3. Send $< MA, \# >$ To $\Gamma^{-1}(\#)$; 4. Send $< PE, \#, \#, NOK >$ To $\Gamma^{-1}(\#)$, $\Gamma(\#)$.

Algorithme 2.3 — Accuser les composants NSO

Dans le cas d'un composant *Faulty SEO*, les actions à entreprendre sont différentes. Il faut constater qu'il est impossible de trouver un composant SEO parmi les descendants d'un objet *Faulty SEO*. De plus, un composant *Faulty SEO* a déjà entrepris les démarches nécessaires afin de rechercher ses ancêtres ne possédant pas d'alibi. Il serait donc inutile d'amorcer à nouveau cette démarche. Par contre, une information très utile peut être déduite par le composant *Faulty SEO*. En effet, il est clair qu'aucun de ses descendants ne peut expliquer la manifestation de son alarme. Il identifie ainsi un alibi pour tous ses descendants (algorithme 2.4).

Object Type	Faulty SEO
Event	Receive <MA, x>
Action	1. Send $< ID, \# >$ To $\Gamma(\#)$.

Algorithme 2.4 — Accuser les noeuds Faulty SEO

Lorsqu'un composant reçoit un message de mise en accusation , il entreprend la recherche

d'un alibi. En réalité, il recherche un objet de type SEO. Il existe deux cas où un composant ne doit pas réagir à la réception d'un message de recherche d'alibi. Le premier cas survient lorsqu'un composant de type *Faulty SEO* reçoit ce message. Ce composant doit poser aucune action puisqu'il est impossible pour lui de trouver un descendant de type SEO. En regard du second cas, les composants de type *Faulty NSO*, aucune action doit être entreprise afin de minimiser le nombre de messages échangés dans le système. Puisqu'un composant de type *Faulty NSO* à déjà reçu un message de la forme <MA, x> et qu'il a déjà entrepris la recherche d'un alibi auprès de chacun de ses successeurs, il faut simplement ignorer les messages de la forme <RA, x> lui parvenant.

Les deux types de composants devant réagir au message <RA, x> sont donc les noeuds NSO et SEO. Puisque chacun de ses objets possède des capacités différentes pour identifier un ensemble de composants non fautifs, ils réagiront différemment. Un objets de type NSO ne pouvant pas conclure que ses ancêtres sont non fautifs, il devra s'en remettre à ses descendants. Par contre, un objet SEO identifie clairement qu'aucune situation d'exploitation inacceptable n'est présente à ce noeud et par conséquent, tous ses ancêtres ne peuvent pas être fautifs. Les deux situations précédentes sont présentées par l'algorithme 2.5 et par l'algorithme 2.6.

Object Type	NSO
Event	Receive <RA, x>
Action	1. Send $< RA, \# >$ To $\Gamma(\#)$.

Algorithme 2.5 — Rechercher un alibi pour les composants NSO

Lorsqu'un objet de type SEO reçoit le message <RA, x>, il entreprend une procédure permettant à tous ses ancêtres de conclure à leur innocence. Trois types de composants (NSO, *Faulty NSO* et SEO) peuvent être rencontrés au cours de ce traitement. Les objets

Object Type	SEO
Event	Receive <RA, x>
Action	1. Send $< IA, \# >$ To $\Gamma^{-1}(\#)$.

Algorithme 2.6 — Rechercher un alibi pour les composants SEO

SEO ne doivent pas réagir puisqu'il est clair qu'il auront déjà posé une action à l'égard d'une sollicitation antérieure. Par contre, les composants NSO et *Faulty NSO* réagiront en respect de l'algorithme 2.7.

Object Type	NSO, Faulty NSO
Event	Receive <IA, x>
Action	1. $TAB_{\#,\#}$.état = OK; 2. Send $< IA, \# >$ To $\Gamma^{-1}(\#)$; 3. Send $< PE, \#, \#, OK >$ To $\Gamma^{-1}(\#)$, $\Gamma(\#)$.

Algorithme 2.7 — Innocenter les ancêtres

L'algorithme 2.7 a présenté une première méthode pour identifier les composants non fautifs. Toutefois, il existe une seconde possibilité pour identifier de tels composants, le message <ID, x>. La réception de ce message signifie qu'il existe un ancêtre du noeud récepteur qui est de type *Faulty SEO*. Puisque la rencontre d'un composant *Faulty SEO* permet de conclure qu'une faute s'est produite soit à ce composant ou à l'un de ses ancêtres il faut déduire que tous ses descendants ne sont pas fautifs. La marche à suivre dans ce cas est présentée par l'algorithme 2.8.

Le dernier algorithme vise à propager les changements d'états des composants. Afin de minimiser les messages propagés, il est préférable de rediriger seulement les messages impliquant un changement d'état. L'algorithme 2.9 présente la procédure quelque soit la

Object Type	NSO, Faulty NSO, Faulty SEO
Event	Receive <ID, x>
Action	1. $TAB_{\#,\#}$.état = OK; 2. Send $< ID, \# >$ To $\Gamma(\#)$; 3. Send $< PE, \#, \#, OK >$ To $\Gamma^{-1}(\#)$, $\Gamma(\#)$.

Algorithme 2.8 — Innocenter les descendants

nature du composant qui reçoit le message de propagation.

Object Type	NSO, SEO, Faulty NSO, Faulty SEO
Event	Receive <PE, y, x, état>
Action	1. Si $TAB_{\#,y}$.état $\neq$ état Si $TAB_{\#,y}$.ring = Vrai Si état = NOK; $TAB_{\#,y}$.ringing = Vrai $TAB_{\#,y}$.état = état Send <PE, y, x, état> To $\Gamma^{-1}(\#)$, $\Gamma(\#)$ - x.

Algorithme 2.9 — Propager les changements d'états

Maintenant que l'algorithme étendre P_S est complété, suffisamment de données sont accessibles afin d'établir, de manière locale, l'ensemble $A_{R'}$ et l'ensemble P_S. Puisque les messages propagés sont consistants, il est certain que chacun des composants obtiendra exactement les mêmes états pour chacun de tous les autres éléments du système. Afin d'identifier les composants qui expliquent la totalité des éléments de l'ensemble $A_{R'}$ la section 2.2.2 présentera l'algorithme restreindre P_S.

2.2.2 Algorithme Restreindre P_S

Après avoir exécuté successivement l'algorithme Étendre P_S, le calcul de $A_{R\prime}$ et le calcul de P_S, nous obtenons un ensemble de composants incapables de se trouver un alibi par les trois premiers cas de la section 2.1. Puisqu'il est suffisant pour un noeud de l'ensemble P_S de couvrir tous les composants contenus dans l'ensemble $A_{R\prime}$ pour demeurer un fautif potentiel ou de ne pas couvrir tous les composants membres de $A_{R\prime}$ pour être déclaré non fautif, il est possible de construire un algorithme autour de ces deux concepts.

Pour réaliser cet algorithme, deux approches peuvent être utilisées : une ascendante et l'autre descendante. La méthode ascendante démarre des composants de l'ensemble $A_{R\prime}$ et envoie des messages vers tous leurs prédécesseurs. Lorsqu'un noeud reçoit un message, il se présente deux alternatives: si le noeud est un élément de P_S, il faut compter le nombre de messages de source différente qu'il reçoit et expédier à nouveau une recherche de couverture vers les composants prédécesseurs. Un composant qui reçoit un message de chacun des éléments de l'ensemble $A_{R\prime}$ demeure un candidat pouvant expliquer la faute. Dans le cas contraire, le noeud possède un alibi.

En ce qui concerne la méthode descendante, elle est plus complexe et nécessite plus d'étapes. Plutôt que de commencer par les noeuds de l'ensemble $A_{R\prime}$, il faut amorcer le processus à partir des composants de l'ensemble P_S. Cette méthode est plus complexe puisqu'elle nécessite tout d'abord de parcourir les descendants d'un noeud à la recherche d'une alarme active. Après avoir trouvé un tel composant, il faut indiquer à tous les prédécesseurs de ce noeud que ce dernier couvre l'alarme trouvée. Par souci de minimiser le nombre de messages et le nombre d'étapes, l'approche ascendante sera privilégiée.

Afin de réaliser l'algorithme en utilisant une approche ascendante, on remarque qu'il est nécessaire de comptabiliser le nombre de messages reçu de la part des composants ayant une alarme active. Il faut donc introduire une nouvelle structure de données nommée $Compteur_{i,j}$ avec $i \in A_{R\prime}$ et $j \in P_S$. Ainsi, chaque composant $\in P_S$ maintiendra i compteurs correspondant aux noeuds retrouvés dans l'ensemble $A_{R\prime}$. L'interprétation suivante sera donnée à ces compteurs : $Compteur_{i,j}$ désigne le nombre de messages reçu au noeud j pour le composant i (le noeud i possède une alarme bruyante). Il faut aussi remarquer que pour un compteur donné ($Compteur_{i,j}$, par exemple), il y a plusieurs valeurs possibles. Une valeur de zéro indique qu'il n'y a pas de chemin à partir du noeud j vers le composant i (i possède une alarme active). Par contre, une valeur supérieure à zéro indique le nombre de chemins différents pour passer du composant j vers le noeud i.

La figure 2.3 présente des exemples pour des compteurs de valeur zéro, un et deux. Tout d'abord, au sujet de la figure 2.3(A) les conditions suivantes s'appliquent : $A_{R\prime} = \{\text{c, d}\}$ et $P_S = \{\text{a, b}\}$. Pour se conformer à la définition donnée des compteurs, on remarque la présence de quatre compteurs: $Compteur_{c,a}$, $Compteur_{d,a}$, $Compteur_{c,b}$ et $Compteur_{d,b}$. Puisqu'il existe un unique chemin du composant a vers les noeuds c et d, la valeur de deux compteurs présents au noeud a ($Compteur_{c,a}$ et $Compteur_{d,a}$) sera 1. Par contre, au noeud b, $Compteur_{d,b}$ vaudra un alors que $Compteur_{c,b}$ prendra une valeur de zéro puisqu'il n'existe pas de chemin de propagation d'erreurs entre les composants b et c. Dans le cas de la figure 2.3(B), elle sera utilisé afin d'illustrer un compteur de valeur deux. En posant $A_{R\prime} = \{\text{d}\}$ et $P_S = \{\text{a, b, c, d}\}$ chacun des éléments de P_S devra être doté d'un compteur. Ainsi, $Compteur_{d,b} = Compteur_{d,c} = Compteur_{d,d} = 1$ et $Compteur_{d,a} = 2$ puisqu'il existe deux chemins pour se rendre du noeud a vers le composant d (le premier: (a,b) (b,d) et le second: (a,c) (c,d)). Bien qu'un compteur puisse prendre des valeurs supérieures à deux, la figure 2.3 permet de visualiser des exemples. Maintenant que les

structures de données nécessaire afin d'identifier P_S sont toutes connues, il est possible de procéder à la description de l'algorithme Restreindre P_S. Cet algorithme se décompose en deux étapes: une phase d'initialisation qui sera suivie du traitement des messages.

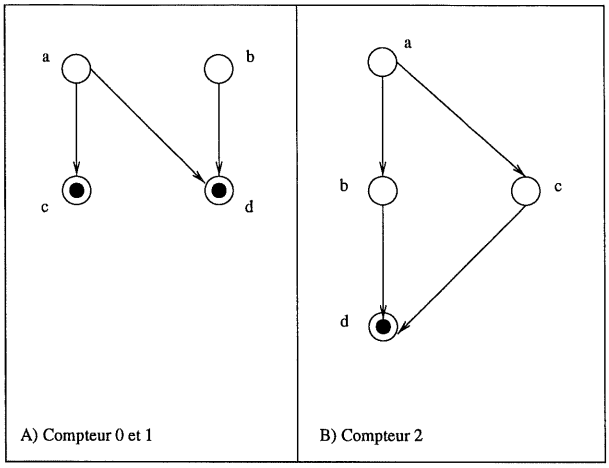

Figure 2.3: *Illustration du compteur*

La phase d'initialisation est utilisée afin de fixer les valeurs de départ de deux structures de données et d'engager le processus de passage des messages. Tout d'abord, il faut mettre l'attribut état de chaque composant à OK. Cette manipulation est rendue nécessaire par le choix de l'implantation de l'algorithme Restreindre P_S. Ainsi, un composant qui devra être retenu potentiellement fautif sera un noeud ayant reçu au moins un message de chacun des composants membres de l'ensemble $A_{R'}$ alors qu'un noeud ne pouvant expliquer la totalité des alarmes de l'ensemble $A_{R'}$ sera identifiable par l'absence de messages provenant d'au moins un composant de l'ensemble $A_{R'}$. Puisqu'il est impossible d'identifier cette absence de message au cours du déroulement de l'algorithme mais qu'il est possible de savoir quel sont les noeuds qui ont une couverture complète des composants de l'ensemble $A_{R'}$, il est obligatoire de procéder à cette initialisation. Une autre structure de données qu'il est nécessaire d'initialiser sont les compteurs. Puisque les composants

potentiellement fautifs seront identifiés à l'aide de la valeur de ces compteurs, leur valeur doit être fixée à zéro avant de débuter l'algorithme. La phase d'initialisation se résume par les actions suivantes:

- $TAB_{i,j}$.State = OK, $\forall$ i $\in$ V, $\forall$ j $\in$ ($A_{R\prime} \cup P_S$);
- $Compteur_{i,j} = 0$, $\forall$ i $\in A_{R\prime}$, $\forall$ j $\in P_S$.

Après avoir réalisé cette initialisation, on remarque que très peu d'information peuvent être obtenues à partir de la structure TAB. Par contre, deux ensembles très utiles sont accessibles : l'ensemble P_S et l'ensemble $A_{R\prime}$. En plus de ces deux ensembles, notre système possède la capacité d'échanger des messages. Les messages nécessaires pour réaliser l'algorithme Restreindre P_S sont présentés par le tableau 2.2.

type	NSO	faulty NSO	SEO	faulty SEO
message				
AR	rien	rien	rien	oui
RC	rien	oui	rien	impossible
PE	oui	oui	oui	oui

Tableau 2.2: *Messages traités par l'algorithme Restreindre P_S*

Avant de poursuivre avec la description de l'algorithme Restreindre P_S, il faut spécifier une hypothèse concernant la synchronisation du système ainsi que certaines définitions. Concernant la synchronisation, il faut supposer qu'à la fin de l'algorithme Étendre P_S, il est possible de calculer $A_{R\prime}$ ainsi que P_S avant de procéder à l'initialisation nécessaire pour le déroulement de l'algorithme Restreindre P_S. Après cette initialisation, il faut doter le système d'un dispositif permettant d'enclencher le déroulement de l'algorithme Restreindre P_S à chacun des composants membres de l'ensemble $A_{R\prime}$. Puisqu'il n'est plus

possible d'établir l'état d'un composant à partir de la structure de données TAB, il faut utiliser les règles suivantes:

- si x est un composant *Faulty NSO* alors $x \in P_S$;
- si x est un composant NSO alors $x \notin P_S$;
- si x est un composant *Faulty SEO* alors $x \in A_{R\prime}$;
- si x est un composant SEO alors $x \notin A_{R\prime}$.

Lors de l'exécution de l'algorithme Restreindre P_S, trois événements peuvent survenir. Le premier événement observable est la réception, pour tous les composants *Faulty SEO*, du message $<$ AR $>$. Ce message identifie le début de l'algorithme Restreindre P_S qui a pour objectif de restreindre l'ensemble P_S aux composants expliquant l'ensemble des alarmes actives ($A_{R\prime}$). L'algorithme 2.10 présente le traitement à réaliser lors de la réception du message $<$ AR $>$. Il faut remarquer que le caractère # désigne le noeud courant.

Object Type	Faulty SEO
Event	Receive $< AR >$
Action	1. Send $< RC, \# >$ To $\Gamma^{-1}(\#)$;
	2. Si $\lvert A_{R\prime}\rvert = 1$
	Send $< PE, \#, \#, NOK >$ To $\Gamma^{-1}(\#)$ $\Gamma(\#)$.
	#.état = NOK.

Algorithme 2.10 — Démarrage de l'algorithme Restreindre P_S

Dans la plupart des cas, l'ensemble P_S contient au moins un élément. Il faut alors vérifier que chacun de ces éléments couvre bien l'ensemble restreint des alarmes bruyantes ($A_{R\prime}$). Pour cette raison, il faut envoyer le message $< RC, \# >$ lors de la présentation de la méthode précédente. Lorsqu'un composant Faulty NSO reçoit un tel message, il doit

vérifier s'il couvre tous les éléments de l'ensemble $A_{R\prime}$. Dans l'éventualité où il peut expliquer l'ensemble $A_{R\prime}$, il sera considéré candidat sérieux pour expliquer la faute présente dans le système (voir algorithme 2.11). Dans le cas contraire, il s'abstiendra de poser un diagnostic concernant son état. Bien entendu, il est impossible qu'un composant *Faulty SEO* reçoive un tel message puisque l'ensemble $A_{R\prime}$ est utilisé pour déterminer l'état d'un composant. De plus, les composants de type NSO et SEO ne réagiront pas à ce message puisque de toute façon, ni eux ni leurs ancêtres ne peuvent se retrouver dans l'ensemble P_S.

Object Type	Faulty NSO
Event	Receive $< RC, x >$
Action	1. Incrémenter $Compteur_{x,\#}$; 2. Send $< RC, x >$ To $\Gamma^{-1}(\#)$; 3. Si $Compteur_{y,\#} > 0 \; \forall$ y $\in A_{R\prime}$ Send $< PE, \#, \#, NOK >$ To $\Gamma^{-1}(\#)$ $\Gamma(\#)$. #.état = NOK.

Algorithme 2.11 — Vérifier la couverture pour les composants Faulty NSO

Le dernier module de l'algorithme (algorithme 2.12) est utilisé afin de propager l'état des composants expliquant $A_{R\prime}$. Quelque soit le type de composant, la réception du message <PE, x, y, état> entraînera toujours les mêmes actions.

Object Type	NSO, SEO, Faulty NSO, Faulty SEO
Event	Receive <PE, x, y, état>
Action	1. Si $TAB_{\#,x}$.état $\neq$ état $TAB_{\#,x}$.état = état Send <PE, x, #, état> To $\Gamma^{-1}(\#)$, $\Gamma(\#)$ - y.

Algorithme 2.12 — Propager les changements d'état

Maintenant que l'ensemble des modules sont présentés, il faut conclure qu'il est possible de réaliser un algorithme distribué afin de poser un diagnostic consistant et valide concernant un élément fautif dans un système. Puisqu'il est possible de réaliser un tel algorithme, la section 2.3 présentera un exemple permettant de mieux visualiser l'échange des messages nécessaires à la réalisation de l'algorithme 2.1.

2.3 Exemple

Dans cette section, on regardera les résultats produits en appliquant l'algorithme présentée à la section 2.2 au graphe illustré par la figure 2.4. Avant qu'une faute se produise, le système est stable et les noeuds présentent tous un état à OK.

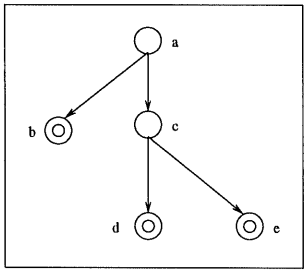

Figure 2.4: *Premier exemple*

À un certain moment, deux alarmes se manifestent et le processus de diagnostic est enclenché par les composants ayant une alarme active, i.e. les noeuds d, e. La figure 2.5 présente la situation après le déclenchement des alarmes.

Premièrement, nous présenterons les messages échangés lors de l'application de l'algorithme Étendre P_S. À la lecture de ces résultats, on remarque une colonne libellée local et une autre intitulée global. La première colonne (TL) désigne l'ordre d'arrivée d'un

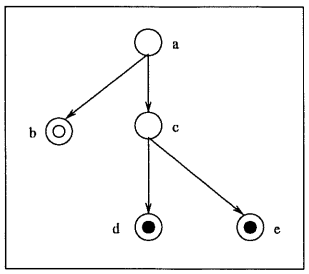

Figure 2.5: *Manifestation d'une faute*

message de manière locale pour chaque composant alors que la seconde colonne représente l'étape générale par rapport au diagnostic dans l'ensemble du système (TG). Par la suite, nous donnerons la structure de données TAB représentant l'état intermédiaire avant l'application de l'algorithme Restreindre P_S (tableau 2.3). Nous terminerons la présentation de l'exemple en appliquant l'algorithme Restreindre P_S et en énonçant le résultat (2.4).

Identification: a		Type: NSO	
Message	Type	Local	Global
$< MA, c >$	NSO	TL1	TG3
$< PE, c, c, NOK >$	FNSO	TL2	TG3
$< PE, d, c, NOK >$	FNSO	TL3	TG3
$< PE, e, c, NOK >$	FNSO	TL3	TG3
$< IA, b >$	NSO	TL3	TG5

Identification: b		Type: SEO	
Message	Type	Local	Global
$< RA, a >$	SEO	TL1	TG4
$< PE, a, a, NOK >$	SEO	TL2	TG4
$< PE, c, a, NOK >$	SEO	TL3	TG4
$< PE, d, a, NOK >$	SEO	TL4	TG4
$< PE, e, a, NOK >$	SEO	TL5	TG4
$< PE, a, a, OK >$	SEO	TL6	TG6

Identification: c		Type: NSO	
Message	Type	Local	Global
$< MA, d >$	NSO	TL1	TG2
$< PE, d, d, NOK >$	FNSO	TL2	TG2
$< MA, e >$	FNSO	TL3	TG2
$< PE, e, e, NOK >$	FNSO	TL4	TG2
$< RA, a >$	FNSO	TL5	TG4
$< PE, a, a, NOK >$	FNSO	TL6	TG4
$< PE, a, a, OK >$	FNSO	TL7	TG6

Identification: d		Type: SEO	
Message	Type	Local	Global
$< NA >$	SEO	TL1	TG1
$< RA, c >$	FSEO	TL2	TG3
$< PE, c, c, NOK >$	FSEO	TL3	TG3
$< PE, e, c, NOK >$	FSEO	TL4	TG3
$< PE, a, c, NOK >$	FSEO	TL5	TG5
$< PE, a, c, OK >$	FSEO	TL6	TG7

Identification: e		Type: SEO	
Message	Type	Local	Global
$< NA >$	SEO	TL1	TG1
$< RA, c >$	FSEO	TL2	TG3
$< PE, c, c, NOK >$	FSEO	TL3	TG3
$< PE, d, c, NOK >$	FSEO	TL4	TG3
$< PE, a, c, NOK >$	FSEO	TL5	TG5
$< PE, a, c, OK >$	FSEO	TL6	TG7

Identification	*Ring*	*Ringing*	état
a	Non	Non	OK
b	Oui	Non	OK
c	Non	Non	NOK
d	Oui	Oui	NOK
e	Oui	Oui	NOK

Tableau 2.3: *Résultats de l'algorithme Étendre* P_S

À partir des informations contenues dans la structure de données TAB (tableau 2.3), il

est maintenant possible d'établir les ensembles A_R, $A_{R\prime}$, et P_S. L'ensemble A_R correspond à tous les composants ayant le champ *ringing* à oui, i.e. les noeuds d et e. L'ensemble $A_{R\prime}$ permet d'identifier les éléments de l'ensemble A_R de premier niveau, i.e. les composants d et e. Afin d'établir le contenu de l'ensemble P_S, il faut regarder la colonne état de la structure de données TAB (tableau 2.3). Dans le cas de l'exemple étudié, $P_S = \{c\}$ puisque les composants d et e ne peuvent s'expliquer mutuellement. Après avoir exécuté les calculs précédents, il est maintenant possible de passer à l'algorithme Restreindre P_S. En appliquant l'algorithme Restreindre P_S, l'échange des messages suivants est obtenu :

Identification: a		Type: SEO	
Message	Type	Local	Global
$< RC, d >$	NSO	TL1	TG3
$< RC, e >$	NSO	TL2	TG3
$< PE, c, c, NOK >$	NSO	TL3	TG3

Identification: b		Type: SEO	
Message	Type	Local	Global
$< PE, c, a, NOK >$	SEO	TL1	TG4

Identification: c		Type: Faulty NSO	
Message	Type	Local	Global
$< RC, d >$	Faulty NSO	TL1	TG2
$< RC, e >$	Faulty NSO	TL2	TG2

Identification: d		Type: Faulty SEO	
Message	Type	Local	Global
$< AR >$	Faulty SEO	TL1	TG1
$< PE, c, c, NOK >$	Faulty SEO	TL2	TG3

Identification: e		Type: Faulty SEO	
Message	Type	Local	Global
$< AR >$	Faulty SEO	TL1	TG1
$< PE, c, c, NOK >$	Faulty SEO	TL2	TG3

À la fin de l'échange des messages, il est possible de conclure, en regardant le tableau 2.4 que le composant c est responsable de la manifestation des alarmes aux noeuds d et e.

Identification	*Ring*	*Ringing*	état
a	Non	Non	OK
b	Oui	Non	OK
c	Non	Non	NOK
d	Oui	Oui	OK
e	Oui	Oui	OK

Tableau 2.4: *Résultats de l'algorithme Restreindre* P_S

CHAPITRE 3

LA RELATION « HAPPENED BEFORE »

Au chapitre précédent, nous avons construit un algorithme afin d'identifier un ensemble de composants permettant d'expliquer la totalité des alarmes actives en considérant la connaissance de très peu d'éléments du système à diagnostiquer. À partir de l'ensemble des prédécesseurs immédiats et de l'ensemble des successeurs immédiats d'un composant, il fallait déduire l'ensemble des fautifs potentiels (P_S). En regardant le nombre de messages nécessaires afin d'arriver à l'identification du composant fautif, il faut admettre qu'il existe sûrement une méthode permettant de diminuer la congestion du réseau de communication entre les composants.

Dans ce chapitre, une nouvelle solution sera apportée en présumant la connaissance d'un ensemble de composants pouvant expliquer chacune des alarmes actives. En disposant, au départ, d'un ensemble de candidats pour chaque composant muni d'une alarme, il devrait être possible de minimiser le temps nécessaire afin d'identifier le composant fautif.

Pour présenter le deuxième algorithme, le chapitre sera organisé de la manière suivante : la première section du chapitre sera consacrée aux différentes définitions supportant l'algorithme présenté à la seconde section alors que la dernière section sera utilisée pour regarder le déroulement d'un exemple.

3.1 Définitions

Dans un système distribué, il est difficile et coûteux d'établir un ordre total permettant d'identifier l'agencement chronologique des événements. Pour contrer ce désavantage, nous proposerons une méthode permettant de calculer a priori l'ensemble des événements suceptibles d'avoir provoqué la manifestation d'une alarme. Ces calculs seront basés sur la relation « happened before » proposée originalement par Lamport [10]. L'équation 3.1 présente cette relation:

a « happened before » b si et seulement si
- a et b sont dans le même processus et a survient avant b

ou

- a est l'envoi d'un message et b est sa réception

ou

- il existe c tel que
  a « happened before » c et
  c « happened before » b (3.1)

Afin d'exprimer cette relation, la notation suivante sera utilisée : $b \nrightarrow a$. En d'autres

termes, a « happened before » b. Lorsque deux événements, a et b, surviennent dans le même processus, c'est qu'il y a une condition inacceptable de fonctionnement (l'événement a) et sa manifestation (l'événement b). Il est clair qu'il est nécessaire qu'une condition inacceptable se produise avant de pouvoir la détecter. Les deux autres composants de la relation font seulement référence à la chaîne de propagation d'une faute.

En bref, la situation peut se décrire ainsi: un composant opérant sous des conditions d'exploitation inadéquates provoquera la manifestation de toutes les alarmes accessibles. De plus, en présence d'une faute simple, il est certain que le composant qui est source de la faute peut être soit:

- un composant muni d'une alarme;
- un composant antécédent d'un composant muni d'une alarme.

À partir des deux dernières constatations, il est possible de construire un ensemble de composants permettant d'expliquer chacune des alarmes du système.

Une première approche permettant de construire de telles ensembles consiste à prendre la totalité des prédécesseurs d'un composant avec alarme. Il faut noter qu'un composant muni d'alarme peut expliquer la manifestation de son alarme. Cet ensemble sera noté K_i (équation 3.2). Pour chacun des composants SEO (composant muni d'une alarme), il existera un ensemble K_i avec $i \in$ A (A étant l'ensemble des composants munis d'une alarme). Ainsi, il aura $\mid A \mid$ ensemble K_i et chacun de ces ensembles contiendra $\mid (\Gamma^{-1})^*(i) \mid + 1$ éléments.

$$K_i = ((\Gamma^{-1})^*(i) \cup \{\mathrm{i}\}) \tag{3.2}$$

À partir des ensembles K_i précédemment définis (équation 3.2), il est possible de tirer deux propriétés importantes. Tout d'abord, chaque ensemble K_i est unique. Cette propriété est garantie par l'hypothèse selon laquelle aucune boucle n'est permise entre les composants du système. La seconde propriété est donnée par l'équation 3.3 et elle permet d'établir un certain ordre en rapport avec deux composants munis d'alarme.

$$K_i \subset K_j \Longleftrightarrow j \succ i \tag{3.3}$$

À partir des ensembles K_i, il est possible d'identifier l'ensemble des fautifs potentiels (l'ensemble P_S). Ainsi, l'équation 3.4 indique la manière de construire l'ensemble P_S. En premier lieu, il faut calculer l'ensemble des composants pouvant contaminer la totalité des éléments de l'ensemble A_R. Ce résultat intermédiaire devra être amputé de tous les noeuds pour lesquels il existe au moins un chemin vers un composant avec une alarme inactive.

$$P_S = \bigcap_{v \in A_R} K_v - \bigcup_{v \in (A - A_R)} K_v. \tag{3.4}$$

En utilisant le cas présenté par la figure 1.3, nous obtenons: $K_g = \{a, b, c, g\}$, $K_h = \{a, c, d, h\}$, $K_m = \{a, b, c, d, f, g, h, i, j, k, m\}$, $A_R = \{g, h, m\}$, $K_f = \{a, , b, f\}$, $K_i = \{a, b, e, f, i\}$, $K_j = \{a, b, f, j\}$ et $A - A_R = \{f, i, j\}$. À partir de ces ensembles, il est possible d'appliquer l'équation 3.4 pour obtenir $P_S = \{c\}$. Le résultat obtenu est tout à fait consistant et cohérent. Par contre, on remarque qu'il y a beaucoup de chevauchements entre certains ensembles.

Les systèmes pouvant être de très grande taille, les chevauchements risquent d'entraîner une explosion du nombre de messages à traiter en plus d'occuper beaucoup d'espace pour emmagasiner chacun des ensemble K_i. À titre d'exemple, la figure 1.3 permet de visuali-

ser des cas de chevauchement. Puisque $m \succ j \succ f$, on remarque que: $K_f \subset K_j \subset K_m$. Le chevauchement étant évident, il faut trouver un moyen pour le réduire sans toutefois alourdir le traitement.

En introduisant les ensembles K_i, nous souhaitions simplifier le traitement nécessaire afin d'identifier l'ensemble des fautifs potentiels (P_S). À prime abord, il serait tentant d'éliminer entièrement le chevauchement entre les ensembles K_i. Intuitivement, les chevauchements correspondent à l'ensemble des composants couverts par chacun des ancêtres SEO d'un noeud SEO. Pour restreindre l'ensemble K_i, il est suffisant de faire l'union de tous les ensembles K_i parmi les ancêtre d'un noeud et de soustraire le résultat obtenu à l'ensemble K_i local.L'équation 3.5 présente l'ensemble obtenu (noté $\overline{K_i}$).

$$\overline{K_i} = K_i - (\bigcup_{v \in ((K_i - \{i\}) \cap A)} (\Gamma^{-1})^*(v) + \{v \in ((K_i - \{i\}) \cap A)\}) \tag{3.5}$$

Une telle action amènerait certaines complications. Nous savons que le cas d'un composant ayant une alarme active permet de conclure que tous les successeurs de ce composant ne peuvent plus être soupçonnés fautifs. Ainsi, il est possible d'établir deux catégories pour les composants de type SEO. L'équation 3.6 présente les composants SEO de premier niveau alors que l'équation 3.7 présente les composants SEO du second niveau.

Un composant v SEO est de premier niveau si et seulement si

$$\nexists \text{ x} \in K_v - \{v\} \text{ tel que x} \in A \tag{3.6}$$

Un composant v SEO est de second niveau si et seulement si

$$\exists \text{ x} \in K_v - \{v\} \text{ tel que x} \in A \tag{3.7}$$

Pour les composants SEO de premier niveau, la construction de l'ensemble $\overline{K_i}$ selon la formule 3.5 n'apporte aucune différence par rapport à l'ensemble K_i. Par contre, en utilisant la formule 3.5 pour les composants SEO de second niveau, une information très

importante est perdue : le niveau du composant.

Au sens du diagnostic, la différence est très simple entre les deux types de composants. Un composant de premier niveau peut affirmer que l'élément fautif fait partie de son ensemble $\overline{K_i}$. Par contre, un noeud de second niveau doit s'assurer qu'aucun de ses prédécesseurs possède une alarme active avant de pouvoir conclure que $\overline{K_i} \subseteq P_S$. En utilisant la restriction stricte sur les chevauchements, les composants devraient établir à quel niveau ils se situent afin d'établir sa stratégie. Pour les composants du second niveau, cette étape est acceptable, par contre, dans le cas des composants du premier niveau, cette étape serait superflue et même inutile.

On constate qu'il est nécessaire de faire un compromis entre la taille des ensembles K_i et la complexité engendrée par une restriction trop stricte de ces ensembles. La complexité introduite provenant de la perte d'information reliée au niveau des composants, il est impératif de conserver cette information. Puisqu'il suffit de regarder les ancêtres d'un composant pour savoir à quel niveau il appartient, il faut s'assurer de conserver les composants munis d'alarme dans les ensembles $\overline{K_i}$. En utilisant la formule 3.8, ce compromis est assuré.

$$\overline{K_i} = K_i - (\bigcup_{v \in ((K_i - \{i\}) \cap A)} (\Gamma^{-1})^*(v)) \tag{3.8}$$

Ainsi, pour chaque composant SEO, une trace de son appartenance au premier ou au second niveau est conservée. L'algorithme présenté à la section 3.2 utilise les ensembles décrits par l'équation 3.8 comme données de base afin de calculer l'ensemble des composants potentiellement fautifs. Il faut noter que le composant fautif se trouve obligatoirement dans les ensembles $\overline{K_i}$ des composants munis d'une alarme active. Par contre, parmi les composants des ensembles $\overline{K_i}$, certains ne peuvent pas être retenus potentiellement

fautifs (ainsi que leurs ancêtres) puisqu'ils possèdent une alarme inactive. L'équation 3.9 présente la manière de calculer l'ensemble P_S.

$$P_S = \bigcap_{v \in A_{R'}} \overline{K_v} - \bigcup_{v \in (A - A_R)} \overline{K_v}. \tag{3.9}$$

3.2 Algorithme

Afin de construire un algorithme distribué à partir des définitions présentées à la section 3.1, il faut apporter certaines extensions au protocole de communication tel que défini à la section 1.2. Le tableau 3.1 présente les modifications apportées à un sous ensemble des messages du protocole original. Ce sous ensemble est composé du groupe minimal de messages nécessaires pour diagnostiquer les origines possibles d'une faute. Pour certains messages, on remarque l'ajout de l'ensemble F_S comme paramètre supplémentaire. Cet ajout est utilisé afin de tirer profit des ensembles $\overline{K_i}$.

Message	Signification
$< NA, m >$	Le composant m a détecté une faute (alarme active).
$< MA, m, F_S >$	Le noeud m accuse l'ensemble F_S d'être fautifs.
$< RA, m, F_S >$	Le noeud m tente de trouver un descendant qui pourra discriminer les élments de F_S.
$< IA, m, F_S >$	Le noeud m avise ses descendants qu'ils ne ne peut rien dire à propos des éléments de F_S.
$< RC, m, F_S >$	Le noeud m veut identifier l'état des composants de l'ensemble F_S .

Tableau 3.1: *Protocole de communication étendu*

Pour amorcer la phase de diagnostic, il faut nécessairement disposer d'un processus permettant à un composant SEO de détecter la présence d'une faute. En résumé, lorsqu'une faute apparait dans le système, un ensemble de composants munis de senseurs détectent la présence de cette faute, manifestent la présence de la faute et envoient le message

approprié (NA) à l'objet relié à ces senseurs. C'est à partir d'ici que le processus de diagnostic distribué est amorcé.

Initialement, seuls les composants SEO muni d'une alarme active peuvent démarrer la phase de diagnostic. Selon le choix d'implantation des ensembles $\overline{K_i}$, il faut séparer le traitement des composants SEO du second et du premier niveau. Ainsi, un composant du second niveau doit reporter à plus tard son diagnostic. Il doit auparavant s'assurer qu'aucun de ses prédécesseurs ne possède une alarme active puisque dans cette éventualité, il n'est pas certain qu'il soit possible de construire un ensemble P_S consistant.

On sait qu'un composant SEO possédant une alarme active peut amener une inconsistance au niveau de l'identification de l'ensemble P_S. On sait aussi qu'aucune inconsistance ne sera introduite si un composant SEO avec une alarme active ne possède pas un prédécesseur SEO manifestant la présence d'une faute. Afin de vérifier cette condition, il faut simplement s'assurer qu'aucun prédécesseur d'un composant SEO muni d'une alarme active ne possède une alarme active. Dans ce cas, le composant SEO se comportera comme un composant SEO de premier niveau. Dans le cas contraire, il ne participera pas au diagnostic. L'algorithme 3.1 présente l'amorce de la phase de diagnostic. Il faut remarquer que le symbole # utilisé dans les algorithmes représentent le composant courant.

À partir du moment où le processus est en marche, le principe est assez simple. Les composants munis d'alarme possèdent des ensembles d'évènements pouvant être survenus antérieurement (les ensembles $\overline{K_i}$). À la section 3.1, l'équation 3.9 fut définie. Cette équation permet de calculer P_S à partir des ensembles $\overline{K_i}$. Il s'agit maintenant d'itérer tant et aussi longtemps que les composants impliqués dans le diagnostic n'arrivent pas à un consensus. La solution recherchée contiendra seulement des composants pouvant

Object Type	SEO
Event	NA
Action	Si $(((\overline{K_{\#}} - \{\#\}) \cap A) = \phi)$ 1. $TAB_{\#,v}$.état = NOK $(\forall\ v \in \overline{K_{\#}})$; 2. $P_S = \overline{K_{\#}}$; 3. Send $< MA, \#, \overline{K_{\#}} >$ To $\Gamma^{-1}(\#)$. Sinon 1. $P_S = \overline{K_{\#}}$; 2. $A_R = ((\overline{K_{\#}} - \{\#\}) \cap A)$ 3. Send $< RC, \#, A_R >$ To $\Gamma^{-1}(\#)$.

Algorithme 3.1 — Notifier les alarmes

expliquer l'ensemble des alarmes du premier niveau.

Dans certains cas, les composants SEO munis d'une alarme active doivent reporter à plus tard leur diagnostic, le temps de s'assurer qu'aucun de leurs prédécesseurs ne possède une alarme active. L'algorithme 3.2 présente le cas pour les composants NSO alors que l'algorithme 3.3 permet d'identifier les traitements pour les composants SEO.

Object Type	NSO
Event	Receive $< RC, x, F_S >$
Action	1. Send $< RC, x, F_S >$ To $\Gamma^{-1}(\#)$.

Algorithme 3.2 — Rechercher couverture pour les composants NSO

Object Type	SEO
Event	Receive $< RC, x, F_S >$
Action	Si $(\# \in F_S)$ 1. Send $< IA, \#, F_S - \# >$ To $\Gamma(\#)$.

Algorithme 3.3 — Rechercher couverture pour les composants SEO

Lors de la recherche de couverture, il faut trouver des composants SEO munis d'alarme inactive. Dès qu'un tel composant est trouvé, il est impératif d'aviser les descendants. Il ne faut pas oublier que les composants *Faulty* SEO doivent s'assurer qu'ils sont du premier niveau avant de pouvoir identifier P_S. La manière d'identifier qu'il existe un composant SEO muni d'une alarme inactive parmi les ancêtres d'un composant est l'envoie du message permettant d'innocenter les ancêtres. L'algorithme 3.4 présente les traitements à réaliser pour les composants NSO et *Faulty* NSO alors que l'algorithme 3.5 permet de traiter le cas des composants SEO.

Object Type	Faulty NSO, NSO
Event	Receive $< IA, x, F_S >$
Action	Send $< IA, x, F_S >$ To $\Gamma(\#)$.

Algorithme 3.4 — Innocenter les ancêtres pour les composants NSO et Faulty NSO

Object Type	Faulty SEO
Event	Receive $< IA, x, F_S >$
Action	1. $A_R = F_S \cap A_R$ Si $(A_R = \phi)$ 1.1. $TAB_{\#,v}$.état = NOK ($\forall\ v \in (\overline{K_\#}$ - A)); 1.2. $P_S = \overline{K_\#} = (\overline{K_\#}$ - A); 1.3. Send $< MA, \#, P_S >$ To $\Gamma^{-1}(\#)$.

Algorithme 3.5 — Innocenter les ancêtres

Maintenant que les modules permettant de s'assurer que seul les composants munis d'alarme du premier niveau sont impliqués dans le diagnostic, regardons les messages permettant d'obtenir l'ensemble P_S. Le premier message qui retiendra l'attention sera le message de mise en accusation. Deux types de composants peuvent le recevoir : les composants NSO et les composants *Faulty* NSO. Dans le cas des composants NSO, le

composant qui reçoit le message de mise en accusation doit changer d'état, accuser ses ancêtres et tenter de se trouver un alibi. L'algorithme 3.6 illustre les traitements à réaliser lors de la réception d'un message de mise en accusation pour un composant NSO. Dans le cas des composants *Faulty* NSO, le message de mise en accusation possède une autre signification. La première action que doit poser un composant *Faulty* NSO, lors de la réception d'un message de mise en accusation, est de s'assurer que l'ensemble F_S permette de modifier l'ensemble P_S. Autrement dit, l'ensemble F_S permet de restreindre l'ensemble P_S. Si l'ensemble P_S est modifié, il faut tenir compte des modifications qu'apportent l'ensemble F_S avant de propager ces changements. Si l'ensemble P_S n'est pas modifié, il suffit d'ignorer le message de mise en accusation. L'algorithme 3.11 présente les détails de la réaction des composants *Faulty* NSO face à la réception d'un message de mise en accusation.

Object Type	NSO
Event	Receive $< MA, x, F_S >$
Action	1. $TAB_{\#,v}$.état = NOK $(\forall\ v \in F_S)$; 2. $P_S = F_S$; 3. Send $< MA, x, F_S >$ To $\Gamma^{-1}(\#)$; 4. Send $< RA, x, F_S >$ To $\Gamma(\#)$.

Algorithme 3.6 — Mettre en accusation pour les composants NSO

Les quatres prochains algorithmes présentés (algorithmes 3.7, 3.8, 3.9, 3.10) concernent le message de recherche d'alibi. Le premier algorithme concernant la recherche d'alibi, l'algorithme 3.7, permet au composant SEO d'innocenter l'ensemble de leur ancêtres. L'algorithme 3.8 permet aux composants *Faulty* SEO de propager les changements de l'ensemble P_S lorsqu'il est nécessaire de modifier cet ensemble.

Les cas les plus simples de recherche d'alibi sont réservés aux composants NSO et *Faulty* NSO. Pour les composants NSO, la seule issue qui se présente est de rechercher un alibi parmi leurs descendants (algorithme 3.9). En ce qui concerne les composants *Faulty* NSO,

Object Type	SEO
Event	Receive $< RA, x, F_S >$
Action	Si $((F_S \cap \overline{K_\#}) \neq \phi)$ 1. $P_S = F_S - \overline{K_\#}$; 2. Send $< MA, \#, P_S >$ To $\Gamma^{-1}(\#)$; Sinon 1. $P_S = F_S$.

Algorithme 3.7 — Rechercher un alibi pour les composants SEO

Object Type	Faulty SEO
Event	Receive $< RA, x, F_S >$
Action	Si $(P_S \neq (F_S \cap \overline{K_\#} \cap P_S))$ et $((\overline{K_\#} - \{\#\}) \cap A) = \phi)$ 1. $TAB_{\#,v}$.état = OK $(\forall\ v \in (P_S - F_S))$; 2. $P_S = F_S \cap P_S$; 3. Send $< MA, \#, P_S >$ To $\Gamma^{-1}(\#)$.

Algorithme 3.8 — Rechercher un alibi pour les composants Faulty SEO

en plus de rechercher un alibi parmi leurs descendants, ils doivent mettre à jour le champ état de la stucture de données TAB pour tous les composants qui ont changés d'état (algorithme 3.10).

Object Type	NSO
Event	Receive $< RA, x, F_S >$
Action	1. Send $< RA, x, F_S >$ To $\Gamma(\#)$.

Algorithme 3.9 — Rechercher un alibi pour les composants NSO

Object Type	Faulty NSO
Event	Receive $< RA, x, F_S >$
Action	Si $(P_S \neq F_S)$ 1. $TAB_{\#,v}$.état = OK $(\forall\ v \in (P_S - F_S))$; 2. $P_S = P_S \cap F_S$; 3. Send $< RA, x, F_S >$ To $\Gamma(\#)$.

Algorithme 3.10 — Rechercher un alibi pour les composants Faulty NSO

Object Type	Faulty NSO
Event	Receive $< MA, x, F_S >$
Action	Si $(P_S \neq F_S)$ 1. $TAB_{\#,v}$.état = OK $(\forall\ v \in (P_S - F_S))$; 2. $P_S = P_S \cap F_S$; 3. Send $< RA, x, P_S >$ To $\Gamma(\#)$; 4. Send $< MA, x, P_S >$ To $\Gamma^{-1}(\#)$.

Algorithme 3.11 — Mettre en accusation pour les composants Faulty NSO

3.3 Exemple

À la section précédente (section 3.2), un ensemble de modules permettant d'identifier l'ensemble P_S furent présentés. Maintenant, cette section présentera le déroulement, étape par étape, de l'algorithme à l'aide d'un exemple. Le système sera composé de six noeuds dont trois composants munis d'alarme. La figure 3.1 présente ce système.

En consultant la figure 3.1, les données suivantes peuvent être extraites : V = { a, b, c, d, e, f }, A = { b, e, f }, A_R = { e, f }, $\overline{K_b}$ = { a, b }, $\overline{K_e}$ = { a, c, e }, $\overline{K_f}$ = { a, c, d, f }. Afin de suivre le déroulement de l'algorithme, on utilisera un tableau pour chaque composant. Ce tableau permettra de voir les messages reçus, l'ensemble P_S, le changement d'état du composant, le temps local (ordre d'arrivée des messages pour un composant) et le temps global. Toutefois, il faut noter que le temps global est ajouté seulement afin de faciliter la compréhension générale, cette mesure n'étant pas disponible en réalité. Le temps global

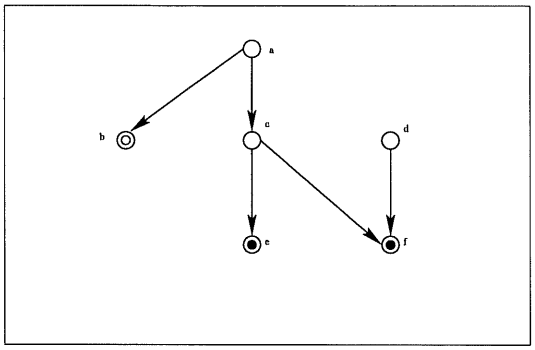

Figure 3.1: *Exemple « happened before »*

permet de visualiser les actions entreprises simultanément dans le système.

Identification: a			Type: NSO	
Message	P_S	Type	Local	Global
$< MA, e, \{a, c, e\} >$	{a, c, e}	Faulty NSO	TL1	TG3
$< MA, f, \{a, c\} >$	{a, c}	Faulty NSO	TL2	TG3
$< MA, b, \{c, e\} >$	{c}	Faulty NSO	TL3	TG5
$< MA, b, \{c\} >$	{c}	Faulty NSO	TL4	TG5
$< MA, b, \{c\} >$	{c}	Faulty NSO	TL5	TG7
$< MA, e, \{c\} >$	{c}	Faulty NSO	TL6	TG9

Identification: b			Type: SEO	
Message	P_S	Type	Local	Global
$< RA, e, \{a, c, e\} >$	{c, e}	SEO	TL1	TG4
$< RA, f, \{a, c\} >$	{c}	SEO	TL2	TG4
$< RA, b, \{c\} >$	{c}	SEO	TL3	TG6

Identification: e			Type: SEO	
Message	P_S	Type	Local	Global
$< NA, e >$	{a, c, e}	Faulty SEO	TL1	TG1
$< RA, e, \{a, c, e\} >$	{a, c, e}	Faulty SEO	TL2	TG3
$< RA, f, \{a, c\} >$	{a, c}	Faulty SEO	TL3	TG3
$< RA, e, \{a, c\} >$	{a, c}	Faulty SEO	TL4	TG5
$< RA, b, \{c\} >$	{c}	Faulty SEO	TL5	TG7
$< RA, e, \{c\} >$	{c}	Faulty SEO	TL6	TG9

Identification: c			Type: NSO	
Message	P_S	Type	Local	Global
$< MA, e, \{a, c, e\} >$	{a, c, e}	Faulty NSO	TL1	TG2
$< MA, f, \{a, c, d, f\} >$	{a, c}	Faulty NSO	TL2	TG2
$< RA, e, \{a, c, e\} >$	{a, c}	Faulty NSO	TL3	TG4
$< MA, f, \{a, c\} >$	{a, c}	Faulty NSO	TL4	TG4
$< MA, e, \{a, c\} >$	{a, c}	Faulty NSO	TL5	TG4
$< RA, f, \{a, c\} >$	{a, c}	Faulty NSO	TL6	TG4
$< RA, b, \{c\} >$	{a, c}	Faulty NSO	TL7	TG6
$< MA, e, \{c\} >$	{c}	Faulty NSO	TL8	TG8
$< MA, e, \{c\} >$	{c}	Faulty NSO	TL9	TG8

Identification: d			Type: NSO	
Message	P_S	Type	Local	Global
$< MA, f, \{a, c, d, f\} >$	{a, c, d, f}	Faulty NSO	TL1	TG2
$< MA, f, \{a, c\} >$	{a, c}	Faulty NSO	TL2	TG4
$< MA, e, \{c\} >$	{c}	Faulty NSO	TL3	TG8
$< RA, e, \{c\} >$	{c}	Faulty NSO	TL4	TG9

Identification: f			Type: SEO	
Message	P_S	Type	Local	Global
$< NA, f >$	{a, c, d, f}	Faulty SEO	TL1	TG1
$< RA, e, \{a, c, e\} >$	{a, c}	Faulty SEO	TL2	TG3
$< RA, f, \{a, c\} >$	{a, c}	Faulty SEO	TL3	TG3
$< RA, f, \{a, c, d, f\} >$	{a, c}	Faulty SEO	TL4	TG3
$< RA, e, \{a, c\} >$	{a, c}	Faulty SEO	TL5	TG5
$< RA, f, \{a, c\} >$	{a, c}	Faulty SEO	TL6	TG5
$< RA, b, \{c\} >$	{c}	Faulty SEO	TL7	TG7
$< RA, e, \{c\} >$	{c}	Faulty SEO	TL8	TG9

En consultant l'ensemble des résultats, on constate que tous les noeuds obtiennent le même diagnostic. Le résultat est donc: P_S = { c }. Le résultat est consistant puisque tous les composants non fautifs sont identifiés comme tels et le noeud fautif est clairement reconnu par tous les composants du système. Bien que l'algorithme présenté dans ce chapitre permette de poser un diagnostic valide et consistant, il possède le désavantage d'utiliser des ensembles statiques ($\overline{K_i}$). Le prochain chapitre posera le problème de diagnostic en utilisant des ensembles dynamiques.

CHAPITRE 4

L'HISTOIRE CAUSALE

En définissant un calcul distribué par un ensemble de processus collaborant pour atteindre un but commun, sans partage de mémoire globale et communiquant par passage de messages, on constate la difficulté d'organiser et de synchroniser les événements. La difficulté d'ordonner les événements provient de la difficulté d'établir une coordination totale entre les processus du système distribué [18]. Puisque dans un système distribué il est très difficile de maintenir et de synchroniser des horloges physiques, il sera nécessaire d'identifier une méthode alternative permettant de synchroniser les événements. Bien que dans le cas de la relation de causalité, Raynal et al. [16] présentent l'utilisation d'horloge logique comme moyen suffisant pour établir l'histoire causale, il ne sera pas nécessaire d'utiliser de tels horloges pour synchroniser les événements survenant dans les systèmes étudiés.

Pour la suite du chapitre, la section 4.1 présentera les définitions soutenant l'histoire causale. La section 4.2 introduira un algorithme permettant de construire l'ensemble P_S en utilisant l'histoire causale et la section 4.3 présentera un exemple.

4.1 Définitions

Avant de définir l'histoire causale, il serait préférable d'établir la terminologie couramment utilisée afin de modéliser la relation de causalité. Soit un système $\mathcal{S}$, composé d'un ensemble de processus asynchrones (les noeuds) noté s_1, s_2, ..., s_i, ..., s_n. Au fur et à mesure que le temps passe, une certaine suite d'événements (e_i^0, e_i^1, ..., e_i^x, e_i^{x+1}, ...) se manifestent au composant s_i. On utilisera h_i pour faire référence à cette séquence d'événements (équation 4.1).

$$h_i = e_i^0, e_i^1, ..., e_i^x, e_i^{x+1}, ... \tag{4.1}$$

La première question qui peut être soulevée est de savoir si h_i est régi par une relation d'ordre total. La réponse à cette question est donnée par Baldy et al. [5]. En se basant sur la relation « happened before » de Lamport [10], Baldy et al. [5] établissent que l'ensemble des événements survenant à un composant répond à un ordre total. Ainsi, h_i est totalement ordonné. Cette propriété sera désignée par la relation d'enchaînement et elle sera notée (h_i, $<_i$). La règle d'enchaînement ($<_i$) exprime la dépendance causale pour le processus s_i. La relation d'ordre total obtenue en appliquant $<_i$ à h_i (équation 4.2) sera notée $\mathcal{H}_i$.

$$\mathcal{H}_i = (h_i, <_i) \tag{4.2}$$

Après avoir établi que les événements survenant localement à un composant sont totalement ordonnés et que seulement l'envoie de messages, la réception de messages et les événements internes peuvent survenir à chacun des noeuds, comment peut-on qualifier

l'ordonnancement des événements survenant à deux composants différents?

Raynal et al [16] présentent la solution en introduisant la relation $<_{msg}$. La relation $<_{msg}$ représente l'ordre établi entre l'envoie et la réception de chaque message en transit dans le système. Ainsi, cette relation définit les dépendances causales entre les événements d'envoi et de réception de messages. La relation d'ordre entre les messages est donnée par l'équation 4.3.

$$send(m) <_{msg} receive(m) \tag{4.3}$$

Pour obtenir une idée sur l'ordre introduit par la relation $<_{msg}$, Baldy et al. [5] apportent l'explication suivante:

> Soient x et y, deux messages provenant de deux processus différents. $x <_{msg} y$ si et seulement si x est l'envoi d'un message et y est la réception du même message par un autre processus.

À partir de l'explication fournie par Baldy et al. [5], on peut constater que la relation $<_{msg}$ définie les dépendances causales entre les événements d'envoie et de réception d'un même message.

À ce point, il est maintenant pertinent de se demander si tous les événements survenus dans le système sont totalement ordonnés. La réponse à cette question est fournie par Raynal et al. [16]. En effet, ils concluent que l'exécution distribuée d'un ensemble de processus est soumis à un ordre partiel. L'équation 4.4 présente la notation utilisée pour exprimer cet ordre.

$$\mathrm{H} = (H, \prec) \tag{4.4}$$

Introduisons à présent l'équation 4.5 comme suit:

$$H = \bigcup_i h_i \tag{4.5}$$

Ainsi, le premier composant de l'ordre H, l'ensemble H (équation 4.5), représente tous les événements survenus dans le système. Intuitivement, l'équation 4.5 exprime simplement le fait que l'histoire causale globale (noté H) est l'union de toutes les histoires causales h_i.

L'histoire causale globale se résume par l'agencement chronologique de deux types d'événements : les événements internes ($<_i$) et les événements externes ($<_{msg}$). Il est maintenant possible d'exprimer l'opérateur de préséance (noté $\prec$) sur H (équation 4.6).

$$\prec = (<_i \cup <_{msg}). \tag{4.6}$$

Une telle relation peut se définir comme suit: soient deux événements *e1* et *e2*. Deux cas peuvent se produire.

1. Si *e1* et *e2* proviennent du même processus alors *e1* $\prec$ *e2* si et seulement si *e1* $<_i$ *e2*.

2. Si *e1* et *e2* ne proviennent pas du même processus alors *e1* $\prec$ *e2* si et seulement si *e1* $<_{msg}$ *e2*.

Pour chaque composant, nous disposons de l'ensemble des événements locaux (h_i). Puisque certains composants ne peuvent pas contaminer certains autres (propagation d'erreur), il est impossible d'ordonner certains événements entre deux processus différents (processus concurrents). La relation $\prec$ représente donc les dépendances causales de la totalité des événements lors de l'exécution distribuée d'un ensemble de processus.

Puisque l'histoire causale représente seulement un ordre partiel, certains événements ne pourront pas être ordonnés. Au fait, l'histoire causale amène trois cas:

1. $e_1 \prec e_2$
2. $e_2 \prec e_1$
3. $e_1 \parallel e_2$

Les deux premiers cas expriment la possibilité que deux événements soient directement ou transitivement dépendants. Par exemple, $e_1 \prec e_2$ signifie que e_2 est directement ou transitivement dépendant de e_1. En d'autres termes, l'événement e_1 est survenu avant l'événement e_2. Mais, puisqu'il est impossible d'ordonner tous les événements survenus dans le système, nous devons admettre le troisième cas: deux événements peuvent être concurrents. Deux événements seront concurrents si $e_1 \not\prec e_2$ ou $e_2 \not\prec e_1$.

Pour se familiariser avec la relation de causalité, nous utiliserons la figure 4.1. En regardant cette figure, on remarque trois lignes horizontales (*p1, p2, p3*). Il s'agit de l'espace temps de chacun des trois processus d'un système. En plus du déroulement des processus, nous pouvons observer des points représentant les événements survenant dans le système. Certains événements sont internes à un processus (*a, c ,d*) alors que d' autres sont externes (*b, e*). Ainsi, l'événement *a* est un traitement interne du processus *p1* alors que l'événement *b* est l'envoi d'un message du processus *p1* vers le processus *p2* et l'événement *g* est la réception, par le processus *p2*, d'un message provenant du processus *p1*. En appliquant la relation d'enchaînement à chacun des processus de la figure 4.1 nous obtenons l'ordre total $\mathcal{H}_i$. Ainsi, $\mathcal{H}_{p1}$ = {a, b, c, d, e}, $\mathcal{H}_{p2}$ = {f, g, h, i, j, k} et $\mathcal{H}_{p3}$ = {l, m, n, o}. Mais, l'ordre $\mathcal{H}_i$ ne donne pas d'information concernant l'agencement des événements entre plusieurs processus. Afin d'obtenir cette information supplémentaire,

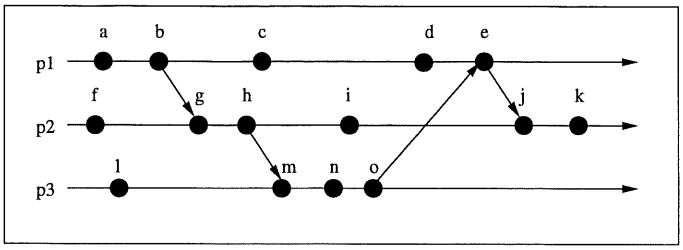

Figure 4.1: *Diagramme d'exécution pour la relation de causalité*

nous utiliserons l'ordre H. En plus de la relation d'ordre $\mathcal{H}_i$, on peut remarquer aussi : $b \prec g$, $h \prec m$, $o \prec e$ et $e \prec j$. Nous obtenons donc l'ordre H = {a, b, f, g, h, l, m, n, o, c, d, e, i, j, k}. Toutefois, il ne faut pas oublier que l'ordre H est seulement un ordre partiel. Pour s'en convaincre, il suffit de regarder, par exemple, la relation qu'il existe entre les composants *a* et *f*. En effet, il est impossible de dire quel événement est survenu avant l'autre donc $a \parallel f$. Maintenant que la relation de causalité est présentée, il est possible de regarder l'éventualité de développer un algorithme utilisant cette relation afin de diagnostiquer les composants fautifs pour un système distribué comme ceux présentés au chapitre 1

4.2 Algorithme

Avant de développer l'algorithme, il faut regarder l'héritage que laisse l'histoire causale, les structures de données qui seront nécessaires, les modifications à apporter au protocole et bien entendu, les grandes étapes qui permettront de réaliser un diagnostic consistant et valide.

En prenant pour acquis que l'ensemble des composants potentiellement fautifs est connu,

comment peut-on utiliser l'histoire causale afin d'identifier le noeud coupable? À cette question, l'équation 4.7 amène une partie de la solution. Le processus de diagnostic étant amorcé par les composants munis d'une alarme active, la construction de l'histoire causale sera amorcée à partir de ces composants. Ainsi, pour chaque candidat, son histoire causale devra contenir au minimum l'ensemble de tous les composants avec une alarme active. Avant de présenter l'équation 4.7 il faut noter une extension apportée à la notation de l'histoire causale. Selon l'équation 4.6 l'histoire causale s'applique entre deux composants. Pour simplifier la notation, il sera possible d'utiliser l'histoire causale entre un composant et un ensemble. Pour interpréter cette modification, il faut comprendre que l'opérateur d'ordre ($\prec$) doit s'appliquer à chacun des éléments de l'ensemble impliqué.

$$H = \{x : x \prec y,\ x \in A_{R\prime} \text{ et } y \in P_S\} \tag{4.7}$$

L'équation 4.7 a présenté la manière de construire l'histoire causale. Maintenant, il faut identifier le composant fautif à partir de l'ensemble H. Pour qu'un composant soit candidat, il doit provoquer toutes les alarmes parmi ses descendants de type SEO. L'équation 4.8 permettra au composant x d'identifier les composants fautifs.

$$\begin{aligned} &\text{Si } x \in \text{C et } H_x = A_{R\prime} \text{ alors Coupable}(y). \\ &\text{où } A_{R\prime} \text{ est l'ensemble des alarmes bruyantes selon le composant } y \end{aligned} \tag{4.8}$$

En ce qui concerne les structures de données, il ne sera pas nécessaire d'utiliser la structure TAB tel que définie précédemment. Plutôt que d'utiliser une seule structure de données, l'algorithme utilisera un ensemble de structure, plus spécialisées, afin d'obtenir plus efficacement un diagnostic. En plus des ensembles bien connus $A_{R\prime}$ et P_S, nous ajouterons l'ensemble H pour recevoir les composants formant l'histoire causale, la liste « To Confirm », l'ensemble « To Process » et la liste de tous les messages de mise en accusation (MA) et de recherche d'alibi (RA) reçue à chaque composant. La liste « To Confirm » sera utilisée afin d'emmagasiner les messages nécessitant une réponse afin d'exécuter le

diagnostic. Au fait, il y a deux messages qui peuvent être qualifiés de messages de requêtes : la mise en accusation et la recherche d'alibi. Ces deux messages représentent le noyau de l'algorithme permettant de construire l'histoire causale et l'ensemble $A_{R/}$. Le dernier ensemble qui doit être introduit, l'ensemble « To Process », sert uniquement à discriminer entre différents composants concurrents pour savoir lequel peut-être retenu fautif. Cet ensemble contient deux éléments: un composant potentiellement fautif et l'ensemble $A_{R/}$ soutenant la candidature du premier élément.

Afin de réaliser le diagnostic, le travail à accomplir sera divisé en trois étapes : identifier la liste des candidats potentiels, construire l'histoire causale et identifier le composant coupable. Il faut noter que les deux premières étapes devront absolument être complétées avant de pouvoir identifier adéquatement le coupable. Pour arriver à réaliser l'ensemble de la tâche, le protocole a dû être modifié. En plus des modifications apportées aux messages existants (tableau 4.1, le message < GS , msg-ID > doit être ajouté. Ce message permet

Message	Signification
$< NA, m >$	Le composant m a détecté une faute (alarme active).
$< MA, m, msg - ID >$	Le noeud m accuse ses prédécesseurs d'être fautifs.
$< RA, m, msg - ID >$	Le noeud m tente de trouver un descendant qui pourra inniocenter ses ancêtres.
$< IA, m, msg - ID >$	Le noeud m avise ses ancêtres qu'ils ne sont pas coupables.
$< ID, m, msg - ID >$	Le noeud m avise ses descendants qu'ils ne sont pas coupables.
$< GS, msg - ID >$	Le message identifié par msg-ID ne peut rien apporter au diagnostic.
$< PE, x, A_R, NOK >$	Propager l'identification du composant coupable.

Tableau 4.1: *Protocole de communication pour l'histoire causale*

simplement à un noeud de dire qu'il ne peut rien apporter de nouveau au processus de diagnostic en rapport avec le message possédant le numéro d'identification «msg-ID».

Lorsqu'un accusé de réception est expédié, il faut enlever le message demandant l'accusé de réception de la liste des messages reçus. Il est nécessaire puisque les messages de requêtes doivent nécessairement obtenir une réponse avant de pouvoir exécuter adéquatement leur diagnostic. La suite de cette section sera consacrée à la présentation de l'algorithme. Il faut noter que le caractère # représente le noeud courant et que la fonction MSG-ID(message) retourne un numéro d'identification unique pour chaque message.

Lorsqu'une faute se manifeste, les senseurs des objets SEO la détecte et un message de notification d'alarme est généré (NA). Les composants SEO qui ont détecté la faute amorce le processus de diagnostic. Ils tenteront alors de rendre leurs ancêtres coupables pour la faute qu'ils ont détectée. Dans le cas où il n'existe pas d'ancêtre, le composant SEO doit conclure qu'il est coupable ($\in P_S$). L'algorithme 4.1 présente les détails lors de la réception du message de notification d'alarme.

Object Type	SEO
Event	NA
Action	Ajouter # à A_R; Ajouter # à $H_{\#}$; Si $\Gamma^{-1}(\#) = \phi$ Ajouter # à P_S; Ajouter # et A_R à To Process; Send <PE, #, A_R, NOK> To $\Gamma(\#)$, $\Gamma^{-1}(\#)$; Sinon Ajouter MSG-ID<MA, y> à To Confirm $\forall\ y \in \Gamma^{-1}(\#)$; Send <MA, #, $MSG-ID<MA, y>>$ $\forall\ y \in \Gamma^{-1}(\#)$.

Algorithme 4.1 — Notifier les alarmes

Les deux modules suivants présentent le traitement des messages de mise en accusation pour les objets de type SEO et NSO. Dans le cas des objets NSO (algorithme 4.2), la

mise en accusation oblige le composant à tenter de se trouver un alibi par l'envoi du message de recherche d'alibi (RA). De plus, le composant NSO tentera de rendre ses ancêtres responsables de la faute (envoie du message de mise en accusation). Dans le cas d'un composant SEO, il y a deux possibilités d'actions. Si le composant possède une alarme active, il peut conclure que tous ses descendants sont non fautifs. Par contre, si le composant ne possède pas d'alarme active, il ne peut rien apporter de nouveau au diagnostic. L'algorithme 4.3 présente les détails des traitements à réaliser pour les composants SEO recevant un message de mise en accusation.

Object Type	NSO
Event	<MA, x, msg-id>
Action	Ajouter x à A_R; Ajouter x à $H_{\#}$; Ajouter MSG-ID<MA, y> à To Confirm $\forall\ y \in \Gamma^{-1}(\#)$; Ajouter MSG-ID<RA, y> à To Confirm $\forall\ y \in \Gamma(\#)$; Si $\Gamma^{-1}(\#) = \phi$ Send <GS, msg-id> To $\Gamma(\#)$; Send <MA, x, MSG-ID<MA, y>> $\forall\ y \in \Gamma^{-1}(\#)$; Send <RA, x, MSG-ID<RA, y>> $\forall\ y \in \Gamma(\#)$.

Algorithme 4.2 — Mettre en accusation les composants NSO

Object Type	SEO
Event	<MA, x, msg-id>
Action	Si $\# \notin$ ringing Send <GS, msg-id> To $\Gamma(\#)$; Sinon Send <ID, x, msg-id> To $\Gamma(\#)$.

Algorithme 4.3 — Mettre en accusation les composants SEO

Les messages de recherche d'alibi peuvent être reçus par les composants NSO et SEO. Dans le cas des composants NSO, ils ne peuvent rien apporter au diagnostic. Ils devront

tout de même poursuivre la recherche d'alibi parmi leurs descendants (voir l'algorithme 4.4). Par contre, dès qu'un composant SEO reçoit un message de recherche d'alibi, il doit aviser ses ancêtres qu'ils ne sont pas fautifs. Les détails pour les composants SEO sont présentés par l'algorithme 4.5.

Object Type	NSO
Event	<RA, x, msg-id>
Action	Si $\Gamma(\#) = \phi$ Send <GS, msg-id> To $\Gamma^{-1}(\#)$; Sinon Ajouter MSG-ID<RA, y> à To Confirm $\forall\ y \in \Gamma(\#)$; Send <RA, x, MSG-ID<RA, y>> $\forall\ y \in \Gamma(\#)$.

Algorithme 4.4 — Recherche d'un alibi pour les composants NSO

Object Type	SEO
Event	<RA, x, msg-id>
Action	Si $\# \notin$ ringing Send <IA, x, msg-id> To $\Gamma^{-1}(\#)$; Sinon Ajouter x à A_R; Send <GS, msg-id> To $\Gamma^{-1}(\#)$.

Algorithme 4.5 — Rechercher un alibi pour les composants SEO

L'algorithme 4.6 permet d'innocenter les descendants pour les composants SEO et NSO. Cet algorithme est nécessaire lorsqu'il existe un ancêtre muni d'un alarme active. Dans le cas des composants SEO muni d'une alarme inactive, on fera appel à l'algorithme 4.8 lorsqu'un ancêtre recherche un alibi.

Les deux derniers algorithmes qui ne sont pas encore présentés, l'algorithme 4.7 et l'algorithme 4.9, servent respectivement à confirmer la réception d'un message et à propager les changements d'états des composants.

Object Type	SEO, NSO
Event	<ID, x, msg-id>
Action	$H = \phi$; Enlever msg-id de la liste To Confirm; Send <ID, x, MSG-ID<MA, y>> $\forall\ y \in \Gamma(\#)$>.

Algorithme 4.6 — Innocenter les descendants pour tous les composants

Object Type	SEO, NSO
Event	<GS, msg-id>
Action	Enlever msg-id de la liste To Confirm; Si Liste Vide(To Confirm) Confirmer tous les messages pour lesquels aucun accusé de réception n'a été expédié; Si A_R = H et (A_R et H) $\neq \phi$ Send <PE, #, A_R, NOK> To $\Gamma(\#)$, $\Gamma^{-1}(\#)$; Ajouter # et A_R à To Process;

Algorithme 4.7 — Le composant courant fait seulement accusé réception d'un message

Object Type	NSO
Event	<IA, x, msg-id>
Action	H = ϕ; Enlever msg-id de la liste To Confirm; Confirmer toutes les mises en accusation; Send <IA, x, MSG-ID<RA, y>> $\forall\ y \in \Gamma^{-1}(\#)$.

Algorithme 4.8 — Innocenter les ancêtres pour les composants NSO

Object Type	SEO, NSO
Event	<PE, x, A_R, NOK>
Action	Si x et $A_R \notin$ To Process Ajouter x et A_R à To Process; Send <PE, x, A_R, NOK> To $\Gamma(\#)$, $\Gamma^{-1}(\#)$.

Algorithme 4.9 — Propager la liste des fautifs potentiels

4.3 Exemple

Dans cette section, nous regarderons les résultats produits en appliquant l'algorithmes utilisant l'histoire causale au graphe illustré par la figure 4.2. Avant qu'une faute se produise, le système est stable et les noeuds présentent tous un état à OK.

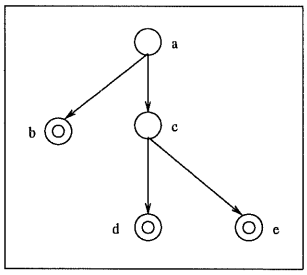

Figure 4.2: *Premier cas*

À un certain moment, deux alarmes se manifestent et le processus de diagnostic est enclenché par les composants ayant une alarme active, i.e. les noeuds *d,e*. La figure 4.3 présente la situation après le déclenchement des alarmes.

Avant de regarder les résultats obtenus, il faut se rappeler que le temps local et le temps global sont utilisés afin de faciliter la compréhension des résultats. Le temps local permet d'ordonner les messages reçus par chaque composant alors que le temps global fait

référence à l'étape de la phase de diagnostic.

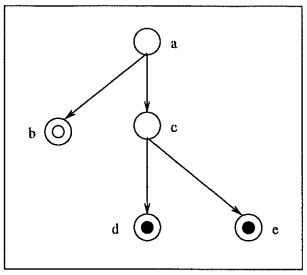

Figure 4.3: *Manifestation d'une faute*

Voici maintenant l'ensemble des messages échangés lors de l'exécution de l'algorithme.

Identification: a		Type: NSO	
Message	Type	Local	Global
$< MA, d, c1 >$	NSO	TL1	TG3
$< MA, e, c4 >$	NSO	TL2	TG3
$< IA, d, a1 >$	NSO	TL3	TG5
$< IA, e, a3 >$	NSO	TL4	TG5
$< GS, a2 >$	NSO	TL5	TG7
$< GS, a4 >$	NSO	TL6	TG7
$< PE, c, \{d, e\}, NOK >$	NSO	TL7	TG7

Identification: b		Type: SEO	
Message	Type	Local	Global
$< RA, d, a1 >$	SEO	TL1	TG4
$< GS, c1 >$	SEO	TL2	TG4
$< RA, e, a3 >$	SEO	TL3	TG4
$< GS, c4 >$	SEO	TL4	TG4
$< PE, c, \{d, e\}, NOK >$	SEO	TL5	TG8

Identification: c		Type: NSO	
Message	Type	Local	Global
$< MA, d, d1 >$	NSO	TL1	TG2
$< MA, e, e1 >$	NSO	TL2	TG2
$< RA, d, a2 >$	NSO	TL3	TG4
$< GS, c1 >$	NSO	TL4	TG4
$< RA, e, a4 >$	NSO	TL5	TG4
$< GS, c4 >$	NSO	TL6	TG4
$< GS, c2 >$	NSO	TL7	TG4
$< GS, c5 >$	NSO	TL8	TG4
$< GS, c3 >$	NSO	TL9	TG4
$< GS, c6 >$	NSO	TL10	TG4
$< GS, c7 >$	NSO	TL11	TG6
$< GS, c9 >$	NSO	TL12	TG6
$< GS, c8 >$	NSO	TL13	TG6
$< GS, c10 >$	NSO	TL14	TG6

Identification: d		Type: SEO	
Message	Type	Local	Global
$< NA >$	SEO	TL1	TG1
$< RA, d, c2 >$	SEO	TL2	TG3
$< RA, e, c5 >$	SEO	TL3	TG3
$< RA, d, c7 >$	SEO	TL4	TG5
$< RA, e, c9 >$	SEO	TL5	TG5
$< GS, d1 >$	SEO	TL6	TG7
$< GS, e1 >$	SEO	TL7	TG7
$< PE, c, \{d, e\}, NOK >$	SEO	TL8	TG7

Identification: e		Type: SEO	
Message	Type	Local	Global
$< NA >$	SEO	TL1	TG1
$< RA, d, c3 >$	SEO	TL2	TG3
$< RA, e, c6 >$	SEO	TL3	TG3
$< RA, d, c8 >$	SEO	TL4	TG5
$< RA, e, c10 >$	SEO	TL5	TG5
$< GS, d1 >$	SEO	TL6	TG7
$< GS, e1 >$	SEO	TL7	TG7
$< PE, c, \{d, e\}, NOK >$	SEO	TL8	TG7

À la fin de l'algorithme, chaque composant obtient le même résultat : le composant c est

responsable de la manifestation des alarmes aux composants d et e.

CHAPITRE 5

PRÉSENTATION D'UN CAS

Dans ce chapitre, nous regarderons les résultats produits en appliquant les trois algorithmes (solution intuitive, la relation « happened before » et l'histoire causale) au graphe illustré par la figure 5.1. Avant qu'une faute se produise, le système est stable et les noeuds présentent tous un état à OK.

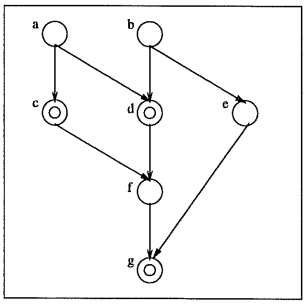

Figure 5.1: *Premier cas*

À un certain moment, deux alarmes se manifestent et le processus de diagnostic est enclenché par les composants ayant une alarme active, i.e. les noeuds *c, g*. La figure 5.2 présente la situation après le déclenchement des alarmes.

Pour la suite du chapitre, la section 5.1 présentera les résultats obtenus en appliquant la solution intuitive, la section 5.2 donnera, pour le même graphe (figure 5.2), l'ensemble des messages échangés pour obtenir un diagnostic valide et consistant alors que la section 5.3 permettra de comparer les résultats obtenus par les deux premiers algorithmes par rapport à l'approche utilisant l'histoire causale.

Avant de regarder les résultats obtenus, il faut se rappeler que le temps local et le temps global sont utilisés afin de faciliter la compréhension des résultats. Le temps local permet d'ordonner les messages reçus par chaque composant alors que le temps global fait référence à l'étape de la phase de diagnostic.

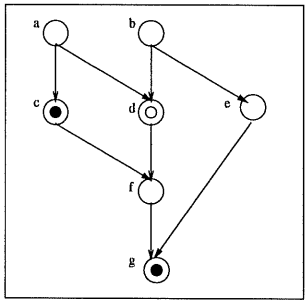

Figure 5.2: *Manifestation d'une faute*

5.1 Résultats en utilisant la solution intuitive

Avant de présenter les résultats, il faut préalablement faire un rappel de la méthode intuitive. Pour qu'un composant soit considéré fautif, il doit respecter deux conditions: ne pas avoir d'alibi et couvrir toutes les alarmes bruyantes. Un composant possédera un alibi dans les cas suivants:

1. Cas 1: x est muni d'une alarme qui ne se manifeste pas;

2. Cas 2: $\exists\ y \in (\Gamma^{-1})^*\ (x)$ tel que y est muni d'une alarme active;

3. Cas 3: $\exists\ y \in \Gamma^*(x)$ tel que y est muni d'une alarme inactive;

L'algorithme est donc divisé en deux phases : les composants doivent se trouver un alibi et par la suite, l'ensemble des composants sans alibi devra être restreint aux composants couvrant toutes les alarmes du système.

Voici les résultat lors de la première étape de l'algorithme intuitif (Étendre P_S).

Identification: a		Type: NSO	
Message	Type	Local	Global
$< MA, c >$	NSO	TL1	TG2
$< PE, c, c, NOK >$	FNSO	TL2	TG2
$< PE, f, c, NOK >$	FNSO	TL3	TG4
$< PE, g, c, NOK >$	FNSO	TL4	TG4
$< IA, d >$	FNSO	TL5	TG4
$< PE, f, d, NOK >$	FNSO	TL6	TG4
$< PE, g, d, NOK >$	FNSO	TL7	TG4
$< IA, d >$	NSO	TL8	TG5
$< PE, b, d, NOK >$	NSO	TL9	TG5
$< PE, e, d, NOK >$	NSO	TL10	TG5
$< PE, e, c, NOK >$	NSO	TL11	TG6
$< PE, f, c, OK >$	NSO	TL12	TG6
$< PE, b, d, OK >$	NSO	TL13	TG6
$< PE, f, d, OK >$	NSO	TL14	TG6
$< PE, g, c, OK >$	NSO	TL15	TG7

Identification: b		Type: SEO	
Message	Type	Local	Global
$< MA, e >$	NSO	TL1	TG3
$< PE, e, e, NOK >$	FNSO	TL2	TG3
$< PE, g, e, NOK >$	FNSO	TL3	TG3
$< IA, d >$	FNSO	TL4	TG4
$< PE, a, d, NOK >$	NSO	TL5	TG4
$< PE, c, d, NOK >$	NSO	TL6	TG4
$< PE, f, d, NOK >$	NSO	TL7	TG4
$< PE, g, d, NOK >$	NSO	TL8	TG4
$< IA, d >$	NSO	TL9	TG5
$< PE, c, e, NOK >$	NSO	TL10	TG5
$< PE, f, e, NOK >$	NSO	TL11	TG5
$< PE, a, d, OK >$	NSO	TL12	TG6
$< PE, f, d, OK >$	NSO	TL13	TG7
$< PE, g, e, OK >$	NSO	TL14	TG7

Identification: c		Type: SEO	
Message	Type	Local	Global
$< NA >$	SEO	TL1	TG1
$< RA, c >$	FSEO	TL2	TG3
$< PE, a, a, NOK >$	FSEO	TL3	TG3
$< MA, f >$	FSEO	TL4	TG3
$< PE, f, f, NOK >$	FSEO	TL5	TG3
$< PE, g, f, NOK >$	FSEO	TL6	TG3
$< PE, a, a, OK >$	FSEO	TL7	TG5
$< PE, e, f, NOK >$	FSEO	TL8	TG5
$< PE, f, f, OK >$	FSEO	TL9	TG5
$< PE, a, a, OK >$	FSEO	TL10	TG6
$< PE, b, a, NOK >$	FSEO	TL11	TG6
$< PE, e, a, NOK >$	FSEO	TL12	TG6
$< PE, b, a, OK >$	FSEO	TL13	TG7
$< PE, g, f, OK >$	FSEO	TL14	TG7

Identification: d		Type: SEO	
Message	Type	Local	Global
$< RA, a >$	SEO	TL1	TG3
$< PE, a, a, NOK >$	SEO	TL2	TG3
$< PE, c, a, NOK >$	SEO	TL3	TG3
$< PE, c, f, NOK >$	SEO	TL4	TG3
$< MA, f >$	SEO	TL5	TG3
$< PE, f, f, NOK >$	SEO	TL6	TG3
$< PE, g, f, NOK >$	SEO	TL7	TG3
$< RA, b >$	SEO	TL8	TG4
$< PE, b, b, NOK >$	SEO	TL9	TG4
$< PE, e, b, NOK >$	SEO	TL10	TG4
$< PE, g, b, NOK >$	SEO	TL11	TG4
$< PE, f, a, NOK >$	SEO	TL12	TG5
$< PE, g, a, NOK >$	SEO	TL13	TG5
$< PE, a, a, OK >$	SEO	TL14	TG5
$< PE, b, b, OK >$	SEO	TL15	TG5
$< PE, g, f, OK >$	SEO	TL16	TG5
$< PE, e, f, NOK >$	SEO	TL17	TG5
$< PE, a, f, NOK >$	SEO	TL18	TG5
$< PE, f, f, OK >$	SEO	TL19	TG5
$< PE, a, a, OK >$	SEO	TL20	TG6
$< PE, f, a, OK >$	SEO	TL21	TG7
$< PE, g, f, OK >$	SEO	TL22	TG7

Identification: e		Type: NSO	
Message	Type	Local	Global
$< MA, g >$	NSO	TL1	TG2
$< PE, g, g, NOK >$	FNSO	TL2	TG2
$< PE, c, g, NOK >$	FNSO	TL3	TG4
$< PE, f, g, NOK >$	FNSO	TL4	TG4
$< RA, b >$	FNSO	TL5	TG4
$< PE, b, b, NOK >$	FNSO	TL6	TG4
$< PE, a, d, NOK >$	FNSO	TL7	TG4
$< PE, b, b, OK >$	FNSO	TL8	TG5
$< PE, a, b, NOK >$	FNSO	TL9	TG5
$< PE, c, b, NOK >$	FNSO	TL10	TG5
$< PE, f, b, NOK >$	FNSO	TL11	TG5
$< PE, g, g, OK >$	FNSO	TL12	TG6
$< PE, f, g, OK >$	FNSO	TL13	TG6
$< PE, a, b, OK >$	FNSO	TL14	TG7
$< PE, f, b, OK >$	FNSO	TL15	TG8

Identification: f		Type: NSO	
Message	Type	Local	Global
$< MA, g >$	NSO	TL1	TG2
$< PE, c, c, NOK >$	FNSO	TL2	TG2
$< PE, g, g, NOK >$	FNSO	TL3	TG2
$< PE, e, g, NOK >$	FNSO	TL4	TG4
$< PE, a, c, NOK >$	FNSO	TL5	TG4
$< ID, c >$	FNSO	TL6	TG4
$< PE, c, d, OK >$	NSO	TL7	TG4
$< PE, b, d, NOK >$	NSO	TL8	TG5
$< PE, e, d, NOK >$	NSO	TL9	TG5
$< PE, a, c, OK >$	NSO	TL10	TG6
$< PE, a, d, OK >$	NSO	TL11	TG6
$< PE, b, d, OK >$	NSO	TL12	TG6
$< PE, g, g, OK >$	NSO	TL13	TG6
$< PE, b, g, OK >$	NSO	TL14	TG7

Identification: g		Type: SEO	
Message	Type	Local	Global
$< NA >$	SEO	TL1	TG1
$< RA, e >$	FSEO	TL2	TG3
$< PE, e, e, NOK >$	FSEO	TL3	TG3
$< PE, c, f, NOK >$	FSEO	TL4	TG3
$< RA, f >$	FSEO	TL5	TG3
$< PE, f, f, NOK >$	FSEO	TL6	TG3
$< PE, b, e, NOK >$	FSEO	TL7	TG5
$< PE, a, e, NOK >$	FSEO	TL8	TG5
$< PE, a, f, NOK >$	FSEO	TL9	TG5
$< ID, f >$	FSEO	TL10	TG5
$< PE, f, f, OK >$	FSEO	TL11	TG5
$< PE, b, e, OK >$	FSEO	TL12	TG6
$< PE, a, e, OK >$	FSEO	TL13	TG8

Identification	Ring	Ringing	état
a	Non	Non	OK
b	Non	Non	OK
c	Oui	Oui	NOK
d	Oui	Non	OK
e	Non	Non	NOK
f	Non	Non	OK
g	Oui	Oui	OK

Tableau 5.1: *Résultats de l'algorithme Étendre* P_S

En regardant les résultats précédents, nous obtenons les valeurs quantitatives présentées par le tableau 5.2. On peut remarquer que la grande majorité des messages échangés sont des messages de propagation de changement d'état. Il serait probablement possible d'optimiser l'algorithme en réduisant le nombre de messages de propagation.

À partir des informations contenues dans la structure de données TAB (tableau 2.3), il est maintenant possible d'établir les ensembles A_R, $A_{R\prime}$, et P_S. L'ensemble A_R correspond à tous les composants ayant le champs ringing à oui, i.e. les noeuds c et g. L'ensemble $A_{R\prime}$

Algorithme Étendre P_S					
Composant	Nombre de message	Nombre d'étape	Message	Nombre	Pourcentage
a	15	7	MA	6	5.61
b	14	7	PE	87	81.3
c	14	7	IA	4	3.74
d	22	7	RA	6	5.61
e	15	8	NA	2	1.87
f	14	7	ID	2	1.87
g	13	8			

Tableau 5.2: *Résultats quantitatifs de l'algorithme Étendre P_S*

permet d'identifier les éléments de l'ensemble A_R de premier niveau, i.e. le composant c. Afin d'établir le contenu de l'ensemble P_S, on doit regarder la colonne état de la structure de données TAB (tableau 2.3). Dans le cas de l'exemple étudié, $P_S = \{c,\ e\}$. Après avoir exécuté les calculs précédents, il est maintenant possible de passer à l'algorithme Restreindre P_S. En appliquant l'algorithme Restreindre P_S, nous obtenons l'échange des messages suivants:

Identification: a		Type: NSO	
Message	Type	Local	Global
$< RC, c >$	NSO	TL1	TG2
$< PE, c, c, NOK >$	NSO	TL2	TG2

Identification: b		Type: NSO	
Message	Type	Local	Global
$< PE, c, d, NOK >$	NSO	TL1	TG4
$< PE, c, e, NOK >$	NSO	TL2	TG5

Identification: c		Type: Faulty SEO	
Message	Type	Local	Global
$< AR >$	Faulty SEO	TL1	TG1

Identification: d		Type: SEO	
Message	Type	Local	Global
$< PE, c, a, NOK >$	SEO	TL1	TG3
$< PE, c, f, NOK >$	SEO	TL2	TG3

Identification: e		Type: Faulty NSO	
Message	Type	Local	Global
$< PE, c, g, NOK >$	Faulty NSO	TL1	TG4
$< PE, c, b, NOK >$	Faulty NSO	TL2	TG5

Identification: f		Type: NSO	
Message	Type	Local	Global
$< PE, c, c, NOK >$	NSO	TL1	TG2
$< PE, c, d, NOK >$	NSO	TL2	TG4

Identification: g		Type: SEO	
Message	Type	Local	Global
$< PE, c, f, NOK >$	SEO	TL1	TG3

Identification	Ring	Ringing	état
a	Non	Non	OK
b	Non	Non	OK
c	Oui	Oui	NOK
d	Oui	Non	OK
e	Non	Non	OK
f	Non	Non	OK
g	Oui	Oui	OK

Tableau 5.3: *Résultats de l'algorithme Restreindre P_S*

Le tableau 5.4 présente les résultats quantitatifs pour l'algorithme Restreindre P_S. Ce qui peut être retenu pour la solution intuitive, c'est qu'elle nécessite treize étapes, qu'il y a eu cent dix neuf messages d'échangés et que quatre vingt deux pourcent des messages sont des messages de propagation d'état. Tout ceci, pour un graphe de sept composants, d'une hauteur de trois et munis de trois alarmes.

Algorithme Restreindre P_S					
Composant	Nombre de message	Nombre d'étape	Message	Nombre	Pourcentage
a	2	2	RC	1	8
b	2	5	PE	10	92
c	1	1	AR	1	8
d	2	3			
e	2	5			
f	2	4			
g	1	3			

Tableau 5.4: *Résultats quantitatifs de l'algorithme Restreindre P_S*

5.2 Résultats en utilisant la relation « happened before »

Avant de présenter les résultats produits par l'algorithme utilisant la relation « happened before », il faut rappeler certaines notions. Tout d'abord, il y a les ensembles $\overline{K_i}$. Ces ensembles contiennent les candidats pouvant expliquer l'alarme au composant i. Il ne faut pas oublier que ces ensembles sont construites de manière statique. Pour construire les ensembles $\overline{K_i}$, nous avons utilisé la relation « happened before ». On peut maintenant regarder les résultats produits par l'application de l'algorithme « happened before ».

Identification: a			Type: NSO	
Message	P_S	Type	Local	Global
$< MA, c, \{a, c\} >$	{a, c}	NSO	TL1	TG2
$< MA, d, \{c\} >$	{c}	Faulty NSO	TL2	TG4
$< MA, c, \{c\} >$	{c}	NSO	TL3	TG6

Identification: b			Type: NSO	
Message	P_S	Type	Local	Global
$< RC, g, \{c, d\} >$		NSO	TL1	TG3
$< MA, d, \{c\} >$	{c}	NSO	TL2	TG4

Identification: c			Type: SEO	
Message	P_S	Type	Local	Global
$< NA >$	{a, c}	SEO	TL1	TG1
$< RA, c, \{a, c\} >$	{a, c}	Faulty SEO	TL2	TG3
$< RC, g, \{c, d\} >$	{a, c}	Faulty SEO	TL3	TG3
$< RA, d, \{c\} >$	{c}	Faulty SEO	TL4	TG5
$< RA, c, \{c\} >$	{c}	Faulty SEO	TL5	TG7

Identification: d			Type: SEO	
Message	P_S	Type	Local	Global
$< RA, c, \{a, c\} >$	{c}	SEO	TL1	TG3
$< RC, g, \{c, d\} >$	{c}	SEO	TL2	TG3
$< RA, d, \{c\} >$	{c}	SEO	TL3	TG5
$< RA, d, \{c\} >$	{c}	SEO	TL4	TG5
$< RA, c, \{c\} >$	{c}	SEO	TL5	TG7

Identification: e			Type: NSO	
Message	P_S	Type	Local	Global
$< RC, g, \{c, d\} >$		NSO	TL1	TG2
$< RA, d, \{c\} >$		NSO	TL2	TG5

Identification: f			Type: NSO	
Message	P_S	Type	Local	Global
$< RC, g, \{c, d\} >$		NSO	TL1	TG2
$< IA, c, \{d\} >$		NSO	TL2	TG4

Identification: g			Type: SEO	
Message	P_S	Type	Local	Global
$< NA >$	{c, d, e, f, g}	SEO	TL1	TG1
$< IA, d, \{c\} >$	{c, d, e, f, g}	Faulty SEO	TL2	TG5
$< RA, d, \{c\} >$	{c}	Faulty SEO	TL3	TG6

En regardant le tableau 5.5, on constate une nette amélioration par rapport à la solution intuitive. Le nombre de messages échangés a chuté de quatre vingt deux pourcent alors que le nombre d'étape a diminué de quarante six pourcent. Il faut se souvenir que le problème majeur avec cette méthode est l'espace occupé par les structures statiques $(\overline{K_i})$.

Algorithme « happened before »					
Composant	Nombre de message	Nombre d'étape	Message	Nombre	Pourcentage
a	3	6	MA	4	18
b	2	4	RA	9	41
c	5	7	NA	2	9
d	5	7	IA	2	9
e	2	5	RC	5	23
f	2	4			
g	3	6			

Tableau 5.5: *Résultats quantitatifs de l'algorithme « happened before »*

5.3 Résultats en utilisant l'histoire causale

À propos de l'histoire causale, il faut se souvenir que l'objectif est de reconstruire l'histoire des composants munis d'une alarme active. À partir de ces composants, on doit identifier quels sont les composants pouvant expliquer la manifestation des alarmes actives. Tout comme dans les deux autres algorithmes, le démarrage s'effectue aux composants munis d'une alarme active. Voici maintenant l'ensemble des messages échangés lors de l'exécution de l'algorithme.

Identification: a		Type: NSO	
Message	Type	Local	Global
$< MA, c, c1 >$	NSO	TL1	TG2
$< PE, c, \{c\}, NOK >$	NSO	TL2	TG4
$< GS, a1 >$	NSO	TL3	TG4
$< IA, c, a2 >$	NSO	TL4	TG4
$< IA, g, b1 >$	NSO	TL5	TG5
$< PE, c, \{c\}, NOK >$	NSO	TL6	TG6

Identification: b		Type: NSO	
Message	Type	Local	Global
$< MA, g, e1 >$	NSO	TL1	TG3
$< IA, c, a2 >$	NSO	TL2	TG4
$< IA, g, b1 >$	NSO	TL3	TG5
$< PE, c, \{c\}, NOK >$	NSO	TL4	TG6
$< PE, c, \{c\}, NOK >$	NSO	TL5	TG7

Identification: c		Type: SEO	
Message	Type	Local	Global
$< NA >$	SEO	TL1	TG1
$< GS, c1 >$	SEO	TL2	TG3
$< RA, c, a1 >$	SEO	TL3	TG3
$< MA, g, f1 >$	NSO	TL4	TG3
$< PE, c, \{c\}, NOK >$	SEO	TL5	TG5
$< PE, c, \{c\}, NOK >$	SEO	TL6	TG5

Identification: d		Type: SEO	
Message	Type	Local	Global
$< GS, c1 >$	SEO	TL1	TG3
$< RA, c, a2 >$	SEO	TL2	TG3
$< MA, g, f2 >$	NSO	TL3	TG3
$< GS, e1 >$	SEO	TL4	TG4
$< RA, g, b1 >$	SEO	TL5	TG4
$< PE, c, \{c\}, NOK >$	SEO	TL6	TG5
$< PE, c, \{c\}, NOK >$	SEO	TL7	TG5
$< PE, c, \{c\}, NOK >$	SEO	TL8	TG7

Identification: e		Type: NSO	
Message	Type	Local	Global
$< MA, g, g1 >$	NSO	TL1	TG2
$< GS, e1 >$	NSO	TL2	TG4
$< RA, g, b2 >$	NSO	TL3	TG4
$< GS, e2 >$	NSO	TL4	TG4
$< GS, f3 >$	NSO	TL5	TG4
$< GS, e3 >$	NSO	TL6	TG6
$< PE, c, \{c\}, NOK >$	NSO	TL7	TG6
$< PE, c, \{c\}, NOK >$	NSO	TL8	TG7

Identification: f		Type: NSO	
Message	Type	Local	Global
$< MA, g, g2 >$	NSO	TL1	TG2
$< PE, c, \{c\}, NOK >$	NSO	TL2	TG4
$< ID, g, f1 >$	NSO	TL3	TG4
$< GS, f2 >$	NSO	TL4	TG4
$< GS, e2 >$	NSO	TL5	TG4
$< GS, f3 >$	NSO	TL6	TG4
$< PE, c, \{c\}, NOK >$	NSO	TL7	TG6
$< GS, e3 >$	NSO	TL8	TG6
$< PE, c, \{c\}, NOK >$	NSO	TL9	TG6

Identification: g		Type: SEO	
Message	Type	Local	Global
$< NA >$	SEO	TL1	TG1
$< RA, g, e2 >$	SEO	TL2	TG3
$< RA, g, f3 >$	SEO	TL3	TG3
$< RA, g, e3 >$	SEO	TL4	TG5
$< PE, c, \{c\}, NOK >$	SEO	TL5	TG5
$< ID, g, c1 >$	SEO	TL6	TG5
$< PE, c, \{c\}, NOK >$	SEO	TL7	TG7

En regardant l'ensemble des messages obtenus, on constate une régression des performances par rapport à l'algorithme « happened before ». Le tableau 5.6 présente un sommaire de la situation.

Algorithme histoire causale					
Composant	Nombre de message	Nombre d'étape	Message	Nombre	Pourcentage
a	6	6	MA	6	12
b	5	7	PE	16	33
c	6	5	GS	12	24.5
d	8	7	IA	4	8
e	8	7	NA	2	4
f	9	6	RA	7	14.5
g	7	7	ID	2	4

Tableau 5.6: *Résultats quantitatifs de l'algorithme histoire causale*

5.4 Comparaison quantitative des résultats

Le tableau 5.7 présente l'ensemble des résultats obtenus par l'application des trois algorithmes.

Algorithme	NA	AR	MA	RA	ID	IA	PE	RC	GS	Total
Solution intuitive	2	1	6	6	2	4	97	1		119
« happend before »	2		4	9		2		5		22
histoire causale	2		6	7	2	4	16		12	49

Tableau 5.7: *Comparaison quantitatives des trois algorithmes*

En regardant les résultats de plus près, on constate que le nombre d'étapes pour l'algorithme utilisant l'histoire causale est comparable à celui de l'algorithme utilisant la relation « happened before ». Par contre, le nombre de messages est plus élevé. Il ne faut pas oublier que l'algorithme utilisant l'histoire causale nécessite que chaque message de mise en accusation et de recherche d'alibi soient confirmés. De plus, l'algorithme utilisant la relation « happened before » profite d'un avantage important: un ensemble de candidats est calculé *a priori* pour chaque alarme pouvant se manifester. Dans le cas de l'exemple, les messages de recherche d'alibi et de confirmation de message (pour l'his-

toire causale) représentent près de soixante dix pourcent de tous les messages. Or, en apportant certaines modifications au protocole de communication, il serait possible de diminuer le nombre de recherche d'alibi et par conséquent, le nombre de confirmations de ces messages. Un composant ne devrait pas avoir besoin d'effectuer une recherche d'alibi pour chaque mise en accusation qu'il reçoit. En éliminant ces chevauchements, les résultats de l'algorithme utilisant l'histoire causale devraient être comparables à ceux de la relation « happend before ».

Regardons à présent le cas de la solution intuitive. Le nombre de message étant considérable, quels sont les améliorations possibles? Le problème majeur de cette solution est indéniablement le nombre de messages de propagation d'état (PE). En retardant le diagnostic avant de propager un changement d'état, il serait possible d'éliminer plusieurs messages de changement d'état. En retardant le diagnostic de culpabilité lors des mises en accusation, le nombre de messages pourraient diminuer grandement.

CONCLUSION

Les systèmes industriels devenant de plus en plus complexes, il devient évident que les approches centralisées utilisées pour le diagnostic des fautes ne sont pas adéquates. Dans ce mémoire, nous avons introduit une approche totalement distribuée dans le but d'arriver un jour à un modèle concret applicable dans la réalité.

Parlant de modèle, il faut maintenant constater que l'utilisation des graphes orientés est appropriée pour représenter les systèmes industriels. Pour s'en convaincre, il suffit de considérer ces systèmes comme étant des systèmes à propagation d'erreurs. Ainsi, une erreur se propagera d'un composant a vers un composant b seulement s'il existe un lien ou une chaîne de liens entre le processus au composant a et le processus au composant b. Après avoir modélisé le système par un graphe, il était aussi nécessaire de le doter d'une capacité de communication et de structures de données. Les structures de données (distribuées) furent utilisées afin d'emmagasiner une vue locale de l'état général du système dans chacun des composants. Pour compléter le modèle, nous avons ajouté des capacités de communication. Il ne faut pas oublier que nous souhaitions obtenir un diagnostic valide et consistant. Par enrichissements successifs, nous avons développé trois approches différentes : une approche intuitive, une approche utilisant la relation « happened before » et une troisième basée sur l'histoire causale.

L'approche intuitive repose uniquement sur une remarque : un composant fautif provoquera la manifestation de toutes les alarmes qui seront dans ses successeurs. De cette approche, on doit retenir ce qui suit :

1. un composant muni d'une alarme inactive permet d'innocenter tous ses antécédents ;

2. un composant sera fautif s'il peut expliquer la manifestation de toutes les alarmes.

Bien que l'approche intuitive fonctionne, elle présente un problème majeur : on doit échanger beaucoup de messages pour arriver à un diagnostic valide et consistant. Puisque le temps de réponse est contraint, un tel algorithme ne peut pas répondre adéquatement à la tâche consistant à identifier le composant fautif.

Puisque l'approche intuitive ne pouvait répondre adéquatement, dans un délai raisonnable, nous avons présenté une seconde solution : la relation « happened before ». Par l'approche intuitive, nous savions qu'un composant fautif doit être en mesure d'expliquer toutes les alarmes actives. Il fallait donc trouver un moyen pour que les composants munis d'alarmes soient en mesure d'identifier les noeuds potentiellement fautifs. Le moyen proposé fut l'utilisation des ensembles $\overline{K_i}$ ($\overline{K_i}$ est l'ensemble des candidats pouvant expliquer l'alarme au composant i). Le problème majeur de l'emploi de tels ensembles réside surtout dans la taille que peuvent prendre les ensembles $\overline{K_i}$. Il ne faut pas oublier que l'on souhaite arriver à une solution applicable à des systèmes industriels complexes qui peuvent être de grande taille. Nous devions donc trouver une solution dynamique.

La solution dynamique que nous avons proposée repose sur l'histoire causale. En utili-

sant cette approche, nous sommes assurés de pouvoir faire évoluer l'algorithme vers une solution acceptable pour les systèmes plus complexes. De par la nature distribuée des systèmes industriels, nous devions disposer d'un mécanisme de synchronisation entre les événements d'un même processus et les événements d'un ensemble de processus. Bien que l'histoire causale ne permette pas d'obtenir un ordre total des événements d'un système, on obtient tout de même un ordre partiel. Cet ordre partiel permet, entre autres, d'identifier les processus dépendants et les processus concurrents en introduisant une notion de temporalité, notion très importante pour les prochaines générations d'algorithmes distribués.

Lorsque l'on parle de la prochaine génération d'algorithmes, il faut être conscient qu'il reste du travail à accomplir avant de pouvoir appliquer les algorithmes à des systèmes industriels concrets. En réalité, il est rare de voir des systèmes où la propagation d'erreur est instantanée. Il sera donc impératif de lever cette restriction. Par la suite, il serait intéressant de voir l'impact de restreindre la capacité de raisonnement seulement aux composants munis d'alarmes (SEO). Après avoir résolu les problèmes précédents, il faudrait pousser l'audace jusqu'à la généralisation de l'algorithme vers la détection des fautes multiples.

BIBLIOGRAPHIE

[1] K.H. Abbott. Robust Operative Diagnosis as Problem Solving in a Hypothesis Space. *Proceedings of the 7th National Conference on Artificial Intelligence*, pages 369–379, 1988.

[2] B. Ayeb. Fault Identification in System-Level Diagnosis: A Logic-Based Framework and An $O(n^2\sqrt{\tau}/\sqrt{logn})$ Algorithm. *Research Report # 218, Submitted*, 1998.

[3] B. Ayeb, J. Ge, and S. Wang. A Unified Model for Abduction Based Reasoning. *IEEE Transactions on Systems, Man and Cybernetics*, 28, July 1998.

[4] B. Ayeb, P. Marquis, and M. Rusinowitch. Preferring Diagnoses By Abduction. *IEEE Transactions on Systems, Man and Cybernetics*, 23:792–808, May 1993.

[5] P. Baldy, H. Dicky, R. Medina, M. Morvan, and J.F. Vilarem. Efficient Reconstruction of the Causal Relationship in Distributed Systems. *Lectures Notes in Computer Science*, vol. 805:pp. 101–113, 1994.

[6] D. Chester, D. Lamb, and P. Dhurjati. Rule-Based Computer Alarm Analysis in Chemical Process Plant. *Proc. Micon-Delcn.*, vol. 22:pp. 22–29, 1984.

[7] R.F. Grove. A Framework For Distributed Diagnostic Reasoning. *International Conference on Parallel and Distributed Computing*, pages pp. 481–486, 1996.

[8] M. Ira, K. Aoki, E. O'shima, and H. Matsuyama. An Algoritm for Diagnosis of System Failures in the Chemical Process. *Comp. Chem. Eng.*, vol. 3:pp. 489–493, 1985.

[9] Y. Ishilda, N. Adachi, and H. Tokumaru. A Topological Approach to Failure Diagnosis of Large Scale Systems. IEEE *Transactions Systems, man, and cybernetics*, vol. SMC-15(no. 3):pp. 327–333, 1985.

[10] L. Lamport. Time Clocks and the Ordering of Events in a Distributed System. *Communications of the ACM*, pages pp. 558–565, 1978.

[11] S.A. Lapp and G.J. Powers. Computer-Aided Synthesis of Fault-Trees. IEEE *Transactions Reliability*, vol. R-26:pp. 2–13, 1977.

[12] Roger S. Pressman. *Software Engineering: a Practitioner Approach.* McGraw-Hill, ISBN: 0-07-050814-3, 1992.

[13] N.S.V. Rao. Computational Complexity Issues in Operative Diagnosis of Graph-Based Systems. IEEE *Transactions on Computers*, vol. 42(no. 4):pp. 447–457, 1993.

[14] N.S.V. Rao. Expected-Value Analysis of two Single-Fault Diagnosis Algorithms. IEEE *Transactions on Computers*, vol. 42(no. 3):pp. 272–280, 1993.

[15] N.S.V. Rao. On Parallel Algorithm for Single-Fault Diagnosis in Fault Propagation Graph System. IEEE *Transactions on Parallel and Distributed Systems*, vol. 7(no. 12):pp. 1217–1223, 1996.

[16] M. Raynal and M. Singhal. Logical Time: Capturing Causality in Distributed Systems. IEEE *Computers Magazine*, vol. 29(no. 2):pp. 49–56, 1996.

[17] J.A. Reggia, D.S. Nau, and P.Y. Wang. Diagnostic Expert Systems Based on a Set Covering Model. *The International Journal of Man-Machine Studies*, vol. 19:pp. 437–460, 1983.

[18] M. Schiper, J. Eggli, and A. Sandoz. A New Algorithm to Implement Causal Ordering. IEEE, pages pp. 219–232, 1985.

[19] J. Shiozaki, H. Matsuyama, E. O'shima, and M. Ira. An Improved Algorithm for Diagnosis of System Failures in the Chemical Process. *Comp. Chem. Eng.*, vol. 9(no. 3):pp. 285–293, 1985.